Titre: Title:	Processeurs embarqués configurables pour la reproduction de tons
Auteur: Author:	Diana Carolina Gil
Date:	2012
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Gil, D. C. (2012). Processeurs embarqués configurables pour la reproduction de tons [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/816/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/816/
Directeurs de recherche: Advisors:	Yvon Savaria, & Pierre Langlois
Programme: Program:	Génie informatique

UNIVERSITÉ DE MONTRÉAL

PROCESSEURS EMBARQUÉS CONFIGURABLES POUR LA REPRODUCTION DE TONS

DIANA CAROLINA GIL

DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE INFORMATIQUE)

AVRIL 2012

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

PROCESSEURS EMBARQUÉS CONFIGURABLES POUR LA REPRODUCTION DE TONS

présenté par : GIL Diana Carolina

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

M. BILODEAU Guillaume-Alexandre, Ph.D., président

M. LANGLOIS J. M. Pierre, Ph.D., membre et directeur de recherche

M. SAVARIA Yvon, Ph.D., membre et codirecteur de recherche

M. BOIS Guy, Ph.D., membre

DÉDICACE

À Alex et à ma famille

REMERCIEMENTS

Je tiens à remercier mon directeur de recherche, le professeur Pierre Langlois, pour m'avoir donné la possibilité de travailler avec lui sur un sujet aussi intéressant et à la fine pointe de la technologie. Sa motivation, son organisation et son inestimable assistance tout au long de ma maîtrise m'ont aidé considérablement à persévérer et à aller plus au-delà de mes barrières imaginaires.

J'aimerais aussi remercier mon co-directeur de recherche, le professeur Yvon Savaria, pour avoir partagé son expertise et m'avoir donné des conseils toujours pertinents. Sa vision et son expérience en tant que chercheur m'ont aidé à effectuer des analyses approfondies sur le sujet en question et la recherche en général.

Je tiens à souligner le support de tous les collègues du laboratoire. En particulier, j'adresse ma gratitude à Shervin Vakili pour sa précieuse collaboration dans la réussite de ces travaux.

Un grand merci à mon mari, Alexander Morante, pour ses constants encouragements, son support intellectuel et sa patience, malgré mes baisses d'humeur. Je remercie aussi ma famille et mes amis pour leur soutien moral, toujours opportun.

J'aimerais également adresser mes remerciements envers l'École Polytechnique de Montréal pour m'avoir accueilli au sein du Département de génie informatique et génie logiciel. De la même façon, le support financier du Fonds québécois de la recherche sur la nature et les technologies (FQRNT) et du Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG) a été grandement apprécié.

Je remercie aussi aux chercheurs qui ont gentiment répondu à mes questions.

Finalement, merci à tous ceux qui, d'une manière ou d'une autre, ont collaboré à la réalisation du présent travail.

¡Mil gracias!

RÉSUMÉ

Les images à grande gamme dynamique (HDR) peuvent capturer les détails d'une scène à la fois dans les zones les plus claires et les zones ombragées, en imitant les capacités du système visuel humain. La reproduction de tons (TM) vise à adapter les images HDR aux dispositifs d'affichage traditionnels.

La première partie de ce travail s'occupe d'une application des algorithmes de reproduction de tons : l'amélioration du contraste. Nous avons effectué une comparaison de plusieurs méthodes de pointe d'ajustement du contraste, y compris deux opérateurs de TM. Cette analyse comparative a été mise en œuvre dans le contexte d'applications de surveillance lorsque les vidéos sont prises dans des conditions d'éclairage faibles. La qualité de l'image a été évaluée en utilisant des métriques objectives comme le contraste d'intensités et l'erreur de la brillance, et via une évaluation subjective. De plus, la performance a été mesurée en fonction du temps d'exécution. Les résultats expérimentaux montrent qu'une technique récente basée sur une modification de l'histogramme présente un meilleur compromis si les deux critères sont considérés.

Les algorithmes de TM imposent habituellement des besoins élevés en ressources de calcul. En conséquence, ces algorithmes sont normalement implémentés sur des processeurs à usage général puissants et des processeurs graphiques. Ces plateformes ne peuvent pas toujours satisfaire les contraintes de performance, de surface, de consommation de puissance et de flexibilité imposées par le domaine des systèmes embarqués. Même si ces exigences sont souvent contradictoires, les processeurs à jeu d'instructions spécialisées (ASIP) deviennent une alternative d'implémentation intéressante. Les ASIP peuvent fournir un compromis entre l'efficacité d'une solution matérielle dédiée et la flexibilité associée à une solution logicielle programmable.

La deuxième partie de ce mémoire présente la conception et l'implémentation d'un processeur spécialisé pour un algorithme global de TM. Nous avons analysé l'algorithme entier afin d'estimer les besoins en données et en calculs. Trois instructions spécialisées ont été proposées : pour calculer les valeurs de la luminance, du logarithme et de la luminance maximale. En utilisant un langage de description architecturale, les instructions spécialisées ont été ajoutées à un processeur similaire à un RISC de 32 bits. Le logarithme a été calculé à l'aide d'une technique spécifique à faible coût basée sur une approximation de Mitchell améliorée. Les résultats

expérimentaux démontrent une augmentation de la performance de 169% si les trois instructions y sont rajoutées, avec un coût matériel supplémentaire de seulement 22%.

Finalement, comme les algorithmes globaux de TM peuvent ne pas préserver d'importants contrastes locaux, nous avons conçu et implémenté un autre ASIP pour un algorithme local. Des instructions spécialisées pour accélérer une pyramide gaussienne modifiée ont été ajoutées à un processeur configurable et extensible, semblable à un RISC de 32 bits. Les différents niveaux de la pyramide ont été calculés en utilisant un noyau gaussien 2D unique dans un processus itératif. Les résultats montrent un facteur d'accélération de 12,3× pour le calcul de la pyramide, ce qui implique une amélioration de la performance de 50% pour l'algorithme local. Ce processeur spécialisé requiert une augmentation de la surface de 19% par rapport à la configuration de base.

ABSTRACT

High dynamic range (HDR) images can capture the details of a scene in both highlights and shadows, imitating the capabilities of the human visual system. Tone mapping (TM) aims to adapt HDR images to conventional display devices.

The first part of this work deals with an application of tone mapping algorithms: contrast enhancement. We compare several state-of-the-art contrast adjustment methods, including two TM operators. This comparative analysis was conducted in the context of surveillance applications when videos are taken in poor lighting conditions. Image quality was evaluated by means of objective metrics such as intensity contrast and brightness error, and by subjective assessment. Moreover, performance was measured based on execution time. Experimental results show that a recent technique based on histogram modification presents a better trade-off considering both aspects.

TM algorithms usually impose high demands on computational resources. As a result, they are usually implemented on powerful general purpose processors and graphics processing units. Such platforms may not meet performance, area, power consumption and flexibility constraints imposed by the embedded system domain. These requirements are often contradictory, and application-specific instruction-set processors (ASIPs) become an interesting implementation alternative. ASIPs can provide a trade-off between the efficiency of a dedicated hardware solution and the flexibility associated with a software programmable solution.

The second part of this master thesis presents the design and implementation of a customized processor for a global TM algorithm. We analyzed the whole algorithm to estimate the data and computational requirements. Three custom instructions were proposed: to calculate luminance, logarithm and maximum luminance values. Using an architecture description language, the custom instructions were added to a 32-bit RISC-based processor. The logarithm was computed using a specific low cost technique based on an improved Mitchell approximation. Experimental results demonstrate a 169% performance improvement when adding all three instructions, with a hardware overhead of only 22%.

Finally, as global TM algorithms may not preserve important local contrasts, we designed and implemented another ASIP for a local algorithm. Custom instructions to accelerate a modified

Gaussian pyramid were added to a configurable and extensible 32-bit RISC-like processor. The different pyramid levels were computed using a unique 2D Gaussian kernel in an iterative process. Results show a speedup factor of 12,3× for the pyramid computation, which implies a 50% performance improvement for the local algorithm. This customized processor requires a 19% area increase compared to the base configuration.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

ADL	Architecture Description Language
ASIC	Application-Specific Integrated-Circuit
ASIP	Application-Specific Instruction-set Processor
CI	Custom Instruction
FPGA	Field-Programmable Gate Array
HDR	High Dynamic Range
LDR	Low Dynamic Range
LISA	Language for Instruction-Set Architecture
PSNR	Peak Signal to Noise Ratio
SIMD	Single-Instruction Multiple-Data
TIE	Tensilica Instruction Extension
TM	Tone Mapping
VLIW	Very Long Instruction Word

LISTE DES ANNEXES

INTRODUCTION

Le rapport entre les régions les plus claires et les plus sombres des scènes du monde réel atteint environ huit ordres de grandeur [1]. Par exemple, la vue à partir d'un poste de pilotage peut inclure le coucher du soleil, la piste d'aviation partiellement ombragée et les parties sombres du tableau de bord. Bien que les appareils photo numériques ordinaires puissent acquérir des images à haute résolution (12 Mpixels et plus), elles ne peuvent pas capturer toute cette plage dynamique avec seulement 8 bits par canal de couleur. Ceci a motivé à étendre la gamme de valeurs que chaque pixel peut représenter. Les images à grande gamme dynamique (HDR, *High Dynamic Range*) viennent combler ce vide en permettant de saisir les détails à la fois dans les zones d'ombre et les zones les plus claires, de manière semblable aux capacités du système visuel humain [1]. En fait, la tendance actuelle est que les caméras à la fine pointe de la technologie incluent un mode de capture HDR, capable de prendre quelques images traditionnelles prises à expositions multiples et de les fusionner. Par la suite, les algorithmes de vision artificielle peuvent traiter ces informations et extraire des caractéristiques impossibles à voir avec l'œil humain.

Cependant, la plupart des technologies d'affichage traditionnelles ne peuvent que reproduire une étroite plage de luminances, d'environ seulement deux ordres de grandeur. Donc, même si l'information capturée par les images HDR peut atteindre des plages de luminance supérieures, par exemple cinq ordres de grandeur, quelques régions des images seront saturées au moment de leur visualisation sur ces dispositifs d'affichage. Afficher les images HDR dans des écrans classiques, tout en préservant le contenu visuel, requiert l'utilisation d'une technique appelée « reproduction de tons » (TM, *tone mapping*) [2]. Cette quête de réduction de la plage dynamique ainsi que du réalisme dans les images a existé depuis l'invention de la photographie [3]. La reproduction de tons a connu un essor effréné cette dernière décennie, étant donné son inclusion dans le domaine de l'infographie et l'intégration de l'imagerie HDR à la photographie numérique.

En effet, la reproduction de tons possède un immense potentiel dans des domaines divers tels que la chirurgie minimalement invasive, la surveillance vidéo, les systèmes de guidage routier et l'assistance dans les postes de pilotage. C'est le cas d'une caméra de surveillance placée de façon à permettre d'observer des parties de la scène à l'intérieur et d'autres à l'extérieur d'un bâtiment.

Ces applications peuvent avoir un impact important sur la sécurité et la santé humaine. Donc, les processeurs embarqués pour reproduction de tons sont nécessaires pour atteindre une ample disponibilité de l'imagerie HDR à la fois efficace et flexible.

Les algorithmes de TM imposent habituellement des besoins élevés en ressources, mais néanmoins ils ont été implémentés en grande partie sur des CPU à usage général et des processeurs graphiques (GPU, *Graphics Processing Unit*) [4, 5]. Dans le domaine des systèmes embarqués en temps réel, les plateformes existantes ne peuvent pas toujours satisfaire les contraintes de surface, de coût, de consommation de puissance et de performance des algorithmes de reproduction de tons. De plus, il n'existe pas de méthode unique qui soit appropriée pour tous les types de scènes, de conditions d'éclairage et d'objectifs [1, 5]. C'est pourquoi un tel système doit aussi offrir une flexibilité élevée [5]. D'ailleurs, on s'attend à ce que les plateformes de développement soient faciles à concevoir pour minimiser le temps de mise en marché autant que possible [6]. Même si ces exigences sont souvent contradictoires, les processeurs à jeu d'instructions spécialisées (ASIP, *Application-Specific Instruction-set Processor*) deviennent une alternative d'implémentation intéressante. Les ASIP peuvent fournir un meilleur compromis entre l'efficacité d'une solution matérielle dédiée et la flexibilité associée à une solution logicielle programmable [7]. De plus, l'effort de conception des ASIP se rapproche à celui d'implémentations logicielles avec un processeur à usage général.

D'un autre côté, plusieurs algorithmes de reproduction de tons sont utilisés pour l'amélioration du contraste d'images à basse gamme dynamique [2]. Pour des applications de surveillance, l'ajustement du contraste en temps réel est une étape requise lorsque les vidéos sont prises dans des conditions de faible éclairage. Un mauvais éclairage peut causer des images à faible contraste, des changements de couleurs et des zones ombragées. Ceci rend difficile l'identification des objets d'intérêt. En fait, il est généralement très difficile de distinguer entre les changements issus de mouvements réels de la cible et les changements dus aux effets de l'ombre et du bruit [8]. De plus, les régions sombres peuvent affecter les résultats de la segmentation en donnant l'impression que la cible est plus grande ou plus petite qu'elle ne l'est en réalité. Il est aussi judicieux de faire ressortir les détails dans les régions ombragées sans introduire des artéfacts. Il est donc important d'évaluer différents algorithmes d'amélioration du contraste en vue de déterminer lequel est le plus performant, compte tenu de la qualité des images et de la complexité.

Ce projet s'inscrit dans la conception de processeurs spécialisés (ASIP) pour l'accélération d'algorithmes de reproduction de tons. Le contexte visé est celui des systèmes embarqués portables à ressources limitées. Nous avons exploré les deux tendances actuelles de conception automatisée d'ASIP : en utilisant un langage de description architecturale et avec un processeur de base configurable et extensible. Plus spécifiquement, les objectifs suivants ont été poursuivis pour cette recherche :

1. Analyser des méthodes de reproduction de tons en termes de la qualité des images résultantes et des besoins en calculs et en mémoire.
2. Implémenter des processeurs spécialisés pour différents types d'algorithmes de reproduction de tons afin d'augmenter leur performance.
3. Développer une architecture pour une technique de filtrage multipasse souvent trouvée dans les algorithmes de reproduction de tons.

Une partie de ces travaux a fait l'objet de deux articles présentés aux conférences ISCAS 2011 [9] et ICECS 2011 [10] de l'IEEE. Le Chapitre 2 ainsi que les deux premières sections du Chapitre 4 sont basés sur ces deux articles.

La contribution principale apportée dans ce travail est l'utilisation de processeurs spécialisés avec des techniques d'extension du jeu d'instructions pour accélérer des algorithmes de reproduction de tons. Ce type d'algorithmes impose habituellement des besoins élevés en calcul. Jusqu'à date, nous n'avons pas trouvé d'approches permettant d'augmenter la performance de ces algorithmes tout en garantissant un compromis avec les contraintes en surface, consommation de puissance et flexibilité, dans le contexte des systèmes embarqués portables à ressources limitées. L'approche ASIP apparait donc un excellent compromis encore inexplorée. Ce projet est un premier pas vers la réalisation d'algorithmes de reproduction de tons en temps réel en utilisant des processeurs spécialisés.

Une contribution additionnelle porte sur l'accélération d'une pyramide gaussienne à l'aide d'instructions spécialisées et d'une architecture SIMD (*Single-Instruction Multiple-Data*). Cette pyramide est essentielle pour le calcul de l'atténuation de gradients de l'algorithme de reproduction de tons de Fattal et al. [2] et peut être adaptée aux autres méthodes de TM ainsi qu'aux diverses techniques du traitement d'images, comme l'analyse de textures et la détection d'arêtes.

La suite du mémoire est organisée en cinq chapitres de la manière suivante :

Le chapitre 1 présente la mise en contexte et la revue de littérature sur l'imagerie HDR et les algorithmes de reproduction de tons, l'amélioration du contraste comme une application de ces algorithmes, les évaluations de la qualité des images et les tendances actuelles pour concevoir des processeurs spécialisés (ASIP).

Le chapitre 2 décrit la conception d'un ASIP pour un algorithme global de reproduction de tons à l'aide du langage de description architecturale LISA. La première section inclut une analyse comparative d'algorithmes d'ajustement du contraste, y compris deux méthodes de TM.

Le chapitre 3 porte sur la conception d'un processeur spécialisé pour un algorithme local de reproduction de tons en utilisant un noyau du processeur configurable et extensible Xtensa.

Le chapitre 4 présente les résultats de ces deux algorithmes de TM, en termes de la qualité des images, de la performance obtenue et de la surface occupée. La première section concerne les résultats d'une analyse expérimentale qui visait à comparer plusieurs méthodes d'amélioration de contraste.

Le chapitre 5 contient une discussion générale sur les algorithmes de reproduction de tons en termes de diverses méthodologies de conception d'ASIP, complexités et métriques de qualité d'images, et les compare avec d'autres travaux trouvés dans la littérature.

Finalement, la conclusion résume les objectifs atteints et donne des indices pour de futurs projets.

CHAPITRE 1 CONTEXTE ET REVUE DE LITTÉRATURE

La reproduction de tons (TM, *tone mapping*) fait partie des techniques utilisées au sein de l'imagerie à grande gamme dynamique (HDR, *High Dynamic Range*). Pour mieux comprendre sa problématique inhérente, nous décrirons les concepts de base de la technologie HDR et ses différences avec l'imagerie classique à la section 1.1. Nous présenterons aussi les types d'algorithmes de reproduction de tons, des techniques multi-échelles et les implémentations existantes. Par la suite, la section 1.2 porte sur une application de ces algorithmes : l'amélioration du contraste des images à basse gamme dynamique (LDR, *Low Dynamic Range*). La section 1.3 contient une revue de métriques de la qualité des images. Finalement, la section 1.4 présente des approches pour concevoir des processeurs à usage spécifique.

1.1 Reproduction de tons

La plage des intensités lumineuses présente dans les scènes du monde réel est de l'ordre de 10^8 :1 [1]. Par exemple, la vue à partir d'un poste de pilotage peut inclure le coucher du soleil, la piste d'aviation partiellement ombragée et les parties sombres du tableau de bord. Dans ces cas, les caméras ordinaires ne réussissent qu'à capturer des images, soit sous-exposées, soit surexposées, entraînant un manque de détails dans les ombres ou une luminosité excessive, respectivement. La Figure 1.1a) représente une scène avec un bon niveau de contraste à l'intérieur d'une maison et une perte presque complète de détails à l'extérieur. Une autre situation peut arriver, comme la Figure 1.1b), dans laquelle la lumière existante n'est pas suffisante pour observer les couleurs de l'image. De plus, lorsqu'une photo est prise de nuit, il est possible que seulement les sources de lumière ou les objets proches d'elles possèdent un rendu utile, tel que montré par la Figure 1.1c). D'ailleurs, un photographe pourrait vouloir exagérer le contraste en vue d'ajouter des effets artistiques, ce qui n'est pas faisable avec une image classique (voir la Figure 1.1d) et la Figure 1.2d)).

L'imagerie à grande gamme dynamique (HDR) permet de capturer des scènes dont la plage de luminosités est proportionnelle à la luminance du monde réel [1]. Ces images peuvent être générées avec des caméras spéciales, ou en utilisant un appareil photo traditionnel et en fusionnant quelques images prises à expositions multiples [2]. Le résultat est donc une seule image contenant un haut niveau de contraste et de détails dans les régions sombres ainsi que dans

les zones les plus claires. La Figure 1.2 contient les images créées avec la technologie HDR correspondantes aux scènes de la Figure 1.1. La Figure 1.2 a) et la Figure 1.2d) ont été obtenues avec le code de l'algorithme de Fattal et al. [2] de la bibliothèque PFStmo [11] . La Figure 1.2b) et la Figure 1.2c) ont été obtenues par l'auteur de [12], à l'aide de la technique d'adaptation locale de Photoshop CS4 et d'un ajustement manuel.

a) Belgium House
Image HDR originale tirée de [13].
Reproduite avec permission.

b) O Canada! No Lights © [12]

c) Golden Gate 2 © [12]

d) Université de Montréal © 2009 Hervé Dang

Figure 1.1: Exemples de scènes capturées avec une caméra normale

En conséquence, les tâches de post-traitement telles que l'ajustement de la couleur et du contraste deviennent plus faciles avec la technologie HDR. L'imagerie médicale, la télédétection, la surveillance vidéo, les systèmes de guidage routier et le cinéma numérique sont quelques domaines qui peuvent en tirer profit, car les valeurs de la luminance acquises peuvent être plus grandes que la plage perceptible par l'œil humain et que celle supportée par certaines technologies d'affichage. En fait, l'œil humain peut s'adapter simultanément à environ 5 ordres de grandeur d'intensité [1], alors que de nombreux dispositifs d'affichage ne peuvent que reproduire une plage dynamique autour de 10^2 :1. Le rendu des images HDR sur des écrans avec un contraste réduit, tout en conservant une correspondance perceptuelle raisonnable à la scène

réelle, requiert une technique spéciale appelée « reproduction de tons » (TM, *tone mapping*). La reproduction de tons consiste donc à compresser la plage dynamique d'une image HDR [1, 4].

a) Belgium house
Image HDR originale tirée de [13].
Reproduite avec permission.

b) O Canada! No Lights © [12]

c) Golden Gate 2 © [12]

d) Université de Montréal © 2009 Hervé Dang

Figure 1.2: Exemples d'images HDR après la reproduction de tons

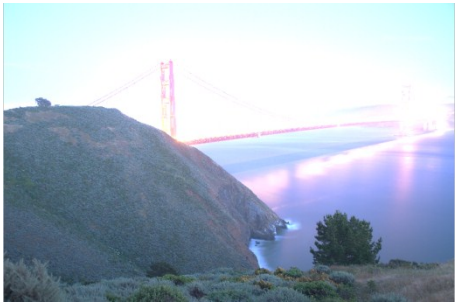
Les images HDR telles quelles sont généralement représentées en virgule flottante avec 32 bits par canal de couleur (96 bits/pixel), alors que la profondeur de bits des images résultantes de la reproduction de tons est de 8 bits par canal (24 bits/pixel) [14]. Les images résultantes sont à basse gamme dynamique (LDR, *Low Dynamic Range*), comme des images traditionnelles. Cependant, les images LDR générées à partir de cette technique contiennent la richesse visuelle propre des images HDR.

L'importance de la reproduction de tons vient du fait que les dispositifs d'affichage ayant une plage dynamique réduite ne peuvent pas afficher adéquatement les valeurs des pixels d'une plage plus élevée. Cela entraîne des pertes de contraste, de détails et des couleurs, justement la même problématique que la technologie HDR veut résoudre. Prenons comme exemple les mêmes scènes de la Figure 1.1. Les images HDR respectives sans reproduction de tons affichées en utilisant des écrans classiques sont montrées à la Figure 1.3, tandis que les images de la Figure

1.2 correspondent aux résultats après la reproduction de tons. C'est pourquoi la reproduction de tons devient une étape cruciale au sein de l'imagerie HDR. Les images de la Figure 1.3 ont été obtenues après avoir affiché directement les valeurs RGB des images HDR originales, avec la fonction *imshow* de MATLAB.

a) Belgium house
Image HDR originale tirée de [13].
Reproduite avec permission.

b) O Canada! No Lights © [12]



c) Golden Gate 2 © [12]

d) Université de Montréal © 2009 Hervé Dang

Figure 1.3: Exemples d'images HDR sans reproduction de tons sur un écran normal

Les sous-sections suivantes portent sur des algorithmes de reproduction de tons, des représentations multi-échelles souvent trouvées dans le domaine de l'imagerie HDR, des études sur la complexité des algorithmes et quelques exemples d'implémentations existantes.

1.1.1 Algorithmes de reproduction de tons

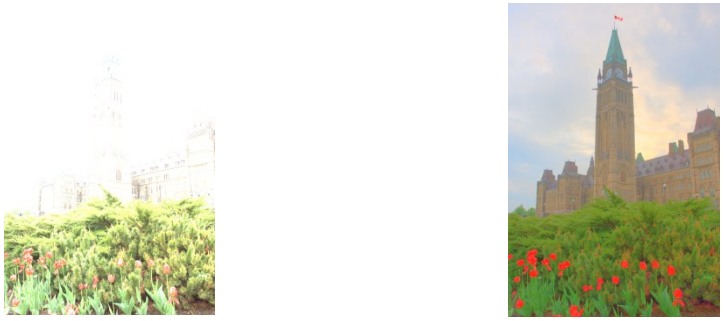
Un grand nombre d'algorithmes de TM ont été développés. Les algorithmes peuvent être catégorisés en deux groupes : globaux et locaux, selon que les pixels de toute l'image soient pris en compte ou seuls les pixels de voisinage, respectivement. Les algorithmes globaux utilisent des transformations spatialement uniformes, basées sur des mesures de l'image dans son ensemble. C'est-à-dire que tous les pixels de l'image adoptent la même fonction de correspondance. Donc, leur principal avantage est l'efficacité computationnelle, au détriment de ne pas préserver

quelquefois d'importants contrastes locaux. Pourtant, les algorithmes locaux utilisent des opérateurs spatialement différents considérant une petite région de pixels voisins. En d'autres termes, différentes transformations sont appliquées selon le contexte spatial local. C'est pourquoi ils présentent une meilleure reproduction des détails, bien qu'ils puissent parfois introduire des artéfacts « halo » [2, 5, 15].

Une première méthode consiste en une simple transformation linéaire d'échelles [16], comme suit :

$$L_d = \frac{L_w - L_{wmin}}{R_w} R_d + L_{dmin}, \tag{1.1}$$

où L_d est la luminance à afficher, L_w représente la luminance réelle, $R_w = (L_{wmax} - L_{wmin})$ et $R_d = (L_{dmax} - L_{dmin})$. Cependant, les résultats avec cette mise à l'échelle naïve ne réussissent pas à reproduire toute la richesse visuelle des scènes réelles [17], car cette approche ne tient pas compte de la densité des valeurs d'intensité et ne fait aucune distinction entre les régions sombres et celles éclairées. Par conséquent, les images résultantes manquent souvent de détails, de contraste et perdent la perception d'ensemble de la luminosité [18]. C'est le cas de l'image montrée à la Figure 1.4a).



a) b)

Figure 1.4: Exemples d'une image HDR reproduite avec les approches les plus simples : a) Après mise à l'échelle linéaire, b) Après mise à l'échelle dans le domaine logarithmique. Image HDR originale « O Canada! No Lights », © [12]

Une deuxième idée consiste à obtenir le logarithme de la luminance entrante et à appliquer, par la suite, la mise à l'échelle linéaire des valeurs du logarithme, comme suit :

$$h = \log(L_w + \delta), \tag{1.2}$$

où δ est une valeur très petite, comme 0,0001;

$$L_d = \frac{h-h_{min}}{R_h} R_d \; + \; L_{dmin}, \tag{1.3}$$

où $R_h = (h_{max} - h_{min})$. Avec cette approche, les résultats sont améliorés en termes de la qualité de l'image dans certains cas, par rapport à la mise à l'échelle sans le logarithme, mais ne sont pas toujours valides pour des images à contraste élevé [17, 19]. Les détails et les textures importants dans les zones les plus claires ou les plus sombres sont normalement détruits avec cette approche [19]. La Figure 1.4b) montre un exemple où les couleurs et les textures ne sont pas affichées adéquatement.

Une troisième approche est d'effectuer la mise à l'échelle avec une fonction sigmoïde, comme celle définie par (1.4) :

$$L_d = \frac{1}{1+\frac{1}{L_w}}. \tag{1.4}$$

De cette façon, les valeurs petites de la luminance seront mises à l'échelle linéairement et les valeurs élevées seront compressées fortement. Cette idée produit de meilleurs résultats et, en fait, plusieurs algorithmes de TM sont basés sur une fonction sigmoïde, car elle prend en compte quelques traits du système visuel humain [1].

L'un des premiers algorithmes de reproductions de tons dans le domaine de l'infographie est celui de Tumblin et Rushmeier [20]. Ils ont développé des modèles mathématiques du système visuel humain en utilisant des données psychophysiques sur la perception de la brillance. Son principe est de trouver un facteur de correspondance de telle sorte que la brillance perçue par le modèle du dispositif d'affichage soit la même que celle du modèle du monde réel. Cette méthode globale est limitée aux images en niveaux de gris, et selon certains auteurs les images résultantes tendent à être trop sombres pour des cas donnés [21].

Par ailleurs, Ward et al. [22] ont présenté un opérateur global linéaire visant la préservation du contraste. Leur approche calcule un facteur d'échelle ayant pour base le modèle psycho-visuel de Blackwell [23]. Cependant, selon les mêmes auteurs, ce facteur n'est pas bien ajusté, puisque les valeurs très élevées et celles très basses sont coupées, ce qui génère des images manquant de visibilité appropriée [18]. Un travail postérieur de Ward et al. [18] a exploité la technique d'égalisation d'histogramme pour préserver la visibilité des objets ainsi que le contraste. Les auteurs ont utilisé le principe que l'œil humain est plus sensible aux différences relatives de la

luminance qu'aux valeurs absolues des différences de la luminance. Ainsi, ce qui intéresse pour afficher une image, c'est que les régions claires soient reproduites par des luminances plus élevées que les régions ombragées, peu importe la différence absolue réelle entre les luminances. De plus, les auteurs ont remarqué que les différents niveaux de luminance dans une scène HDR ne sont pas distribués uniformément sur toute la plage dynamique, mais qu'ils apparaissent en groupes d'intensités diverses. D'abord, leur technique consiste à sous-échantillonner l'image à une résolution correspondant à 1° de l'angle visuel, en imitant l'adaptation locale de l'œil humain. Ensuite, les valeurs de la brillance (la réponse subjective à la lumière) sont approximées en calculant le logarithme des luminances. Par la suite, l'histogramme et la distribution cumulative de l'histogramme du logarithme sont calculés. À partir de la distribution cumulative, les auteurs ont construit une technique d'ajustement d'histogramme, plutôt que d'égalisation, afin de ne pas exagérer le contraste. Finalement, ils ont ajouté des modèles qui considèrent les limitations de la vision humaine, comme la réduction de la sensibilité aux couleurs dans les environnements sombres.

Parmi les algorithmes récents de TM, l'un des plus connus est celui de Reinhard et al. [3]. Cette méthode contient une version globale et une autre locale, inspirées des techniques photographiques. L'approche globale est basée sur la luminance logarithmique moyenne de la scène et une mise à l'échelle linéaire (voir la section 2.2 pour plus de détails). La version locale essaie d'automatiser une technique traditionnelle d'impression de photographies connue comme « maquillage » (*dodging-and-burning*). Son principe consiste à éclaircir une région déterminée de l'image en augmentant la durée d'exposition et à foncer une autre région en diminuant le temps d'exposition. Pour ce faire, les auteurs ont proposé de calculer des profils gaussiens symétriques circulaires à différentes échelles.

Une autre méthode locale très connue dans la littérature est celle de Fattal et al. [2]. L'approche part de la théorie de Horn [24] qui porte sur la séparation de la réflectance de l'éclairement lumineux à l'aide des gradients. L'algorithme de Fattal et al. consiste à atténuer les magnitudes des gradients les plus grands à plusieurs échelles. Les gradients sont associés aux changements drastiques de la luminance. Les auteurs ont calculé ces gradients dans le domaine logarithmique de la luminance, car c'est une approximation des contrastes locaux, auxquels le système visuel humain est plus sensible. Afin de reconstruire l'image de plage dynamique réduite, il est nécessaire de résoudre l'équation de Poisson. À cet effet, Vytla et al. [25] ont proposé une

nouvelle solution directe locale permettant de traiter les données par fenêtres au lieu d'avoir besoin de toute l'image. Pour appliquer cette solution dans le cadre de l'algorithme de Fattal, ils ont dû modifier principalement le calcul de la fonction d'atténuation. Le Chapitre 3 contient plus de détails à ce sujet.

Une autre approche locale a été proposée par Durand et Dorsay [26]. Ces auteurs ont utilisé un filtre de préservation d'arêtes appelé « filtre bilatéral ». Ce filtre non-linéaire est issu du filtre gaussien. Avec le filtre bilatéral, les arêtes ne deviennent pas floues puisque le poids du noyau gaussien est réduit lorsque la différence d'intensités est très grande. Cette méthode décompose l'image HDR en deux couches : une couche de base et une couche de détails. La base correspond à une version lissée de l'image originale à l'aide du filtre bilatéral, et c'est seulement dans cette couche que le contraste est réduit. La visibilité est conservée grâce à la couche de détails.

Par ailleurs, l'opérateur local d'Ashikhmin [27] modélise séparément deux problèmes pertinents du système visuel pour la compression de la plage dynamique : la conservation de la brillance absolue et du contraste local. D'abord, la méthode détermine une valeur d'adaptation locale pour chaque pixel de l'image. La taille de chaque voisinage varie dépendamment du contraste. Ensuite, une fonction non-linéaire est appliquée aux valeurs des luminances d'adaptation afin de réduire la plage dynamique de l'image. Finalement, la méthode réintroduit les détails perdus dans l'étape précédente.

D'autres méthodes à la fine pointe de la technologie incluent celle de Lee et al. [15] qui combine une adaptation globale de la luminance et une adaptation locale basée sur un histogramme par morceaux. Les auteurs ont aussi appliqué des modèles d'adaptation chromatique à la luminance modifiée globalement afin de préserver la chromaticité. D'ailleurs, Duan et al. [28] ont proposé un opérateur global de TM qui travaille entre une quantification linéaire et une égalisation d'histogramme. De plus, cette technique réalise un ajustement adaptatif du contraste en utilisant leur algorithme global dans les régions locales. Shan et al. [29] ont développé une méthode basée sur fenêtres dans laquelle ils résolvent un problème d'optimisation global satisfaisant des contraintes locales. Cette méthode est aussi utilisée pour former une image HDR à partir d'une image LDR unique.

1.1.2 Représentations multi-échelles des images

Les filtres gaussiens et les techniques multi-échelles font souvent partie de nombreuses applications dans le domaine du traitement d'images. Les méthodes de reproduction de tons ne font pas exception à cette règle. C'est le cas des algorithmes de Reinhard [3], Ashikhmin [27], Fattal [2], entre autres. Par exemple, l'atténuation des gradients dans l'algorithme de Fattal repose sur le calcul d'une pyramide gaussienne. Dans ce cas, le but est de détecter les arêtes significatives à plusieurs échelles [2]. D'autres méthodes utilisent aussi une pyramide pour simuler l'adaptation locale du système visuel lors de changements dans le niveau d'éclairage au sein d'une scène HDR [1].

Une pyramide gaussienne est une représentation multi-échelles qui comporte deux étapes : un filtre passe-bas gaussien et un sous-échantillonnage. Le sous-échantillonnage réduit la taille de l'image de façon à obtenir la moitié du nombre de lignes et de colonnes. Ces étapes sont répétées jusqu'à une taille prédéfinie de l'image. Le résultat est donc une pile d'images filtrées à plusieurs résolutions ordonnées d'après la forme d'une pyramide [30], comme l'exemple de la Figure 1.5. La base de la pyramide ou le niveau 0 correspond à l'image originale, sans la filtrer. C'est l'image qui contient tous les détails. Au fur et à mesure que les niveaux augmentent, les fréquences spatiales élevées sont éliminées [30]. À la limite, la valeur moyenne de l'image est obtenue.

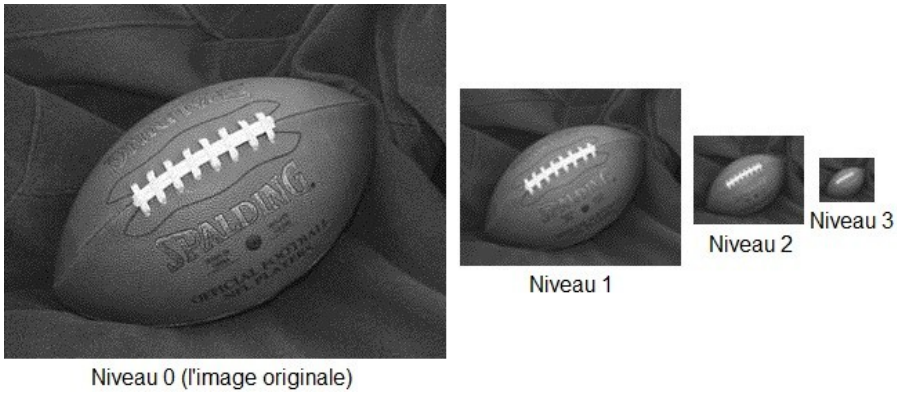


Figure 1.5: Exemple d'une pyramide gaussienne. Image originale prise des exemples de MATLAB © MathWorks.

Un filtre spatial est généralement mis en œuvre via la convolution de l'image avec un noyau de valeurs prédéterminées. La réponse R du filtre linéaire à un point (x, y) de l'image pour un noyau

de $m \times n$ coefficients est définie par (1.5) comme la somme de produits des coefficients du filtre et des valeurs de l'image correspondant à l'aire délimitée par le noyau [30] :

$$R(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t), \quad (1.5)$$

où w représente les poids du noyau, f sont les valeurs d'intensité respectives de l'image, $a = (m-1)/2$, $b = (n-1)/2$, et le centre du noyau $w(0,0)$ doit être placé au point (x,y). Un filtre gaussien est approximé avec un noyau de valeurs suivant la fonction gaussienne définie par (1.6) dans le domaine des fréquences :

$$H(u,v) = e^{-D^2(u,v)/2\sigma^2}, \quad (1.6)$$

où σ est une mesure du taux de décroissance de la fonction gaussienne et $D(u,v)$ est la distance du point (u,v) au centre (l'origine) de la transformée de Fourier [30]. Donc, les coefficients du noyau ont des valeurs plus grandes au centre du noyau et plus petites en périphérie, approximant une forme de courbe en cloche [31].

1.1.3 Complexité et implémentations existantes

Les algorithmes de TM imposent habituellement des besoins élevés en ressources de calcul. En conséquence, les opérateurs de TM ont été implémentés en grande partie sur des CPU à usage général et des processeurs graphiques (GPU, *Graphics Processing Unit*) [4, 5]. L'étude de Reinhard et al. [32] portant sur la complexité des algorithmes de reproduction de tons démontre que ces algorithmes n'atteignent pas une performance suffisante pour des applications en temps réel avec un processeur à usage général. Les auteurs ont testé 20 méthodes différentes sur un iBook Apple avec un processeur G3 à 800 MHz, 512 MB de RAM et des images de 1600×1200 pixels. Selon leurs expériences, le temps d'exécution des algorithmes globaux testés pour une image varie entre 1s et 15s, tandis que celui des algorithmes locaux oscille entre 10s et 2 minutes. Le Tableau 1.1 montre les résultats pour la plupart des algorithmes mentionnés à la sous-section 1.1.1.

Afin de réaliser la reproduction de tons en temps réel, des approches différentes ont été proposées. Par exemple, El-Mahdy et El-Shishiny [4] ont décrit deux méthodes pour accélérer la version locale de l'algorithme photographique proposé par Reinhard et al. [3] : une convolution sélective et une recherche approximée basée sur une technique d'apprentissage machine.

L'analyse théorique montre que l'utilisation de la mémoire est diminuée de 3× par rapport à l'implémentation originale de Reinhard. Les résultats expérimentaux signalent aussi une accélération de 14× par rapport à l'implémentation sur un GPU de Goodnight et al. [33], en utilisant deux processeurs Cell/B.E. Toutefois, les besoins prohibitifs en consommation de puissance de cette plateforme rendraient la reproduction de tons inappropriée pour les systèmes embarqués à ressources limitées.

Tableau 1.1: Temps d'exécution d'algorithmes de reproduction de tons, selon [32]

Type d'algorithme	**Algorithme**	**Temps d'exécution (s)**
Global	Opérateur de Tumblin et Rushmeier	3,2
	Facteur d'échelle de Ward	0,96
	Égalisation d'histogramme de Ward	3,4
	Méthode photographique de Reinhard - global	3,7
Local	Méthode photographique de Reinhard - local	80
	Compression de gradients de Fattal	45,5
	Filtre bilatéral de Durand et Dorsay	23,5
	Opérateur d'Ashikhmin	120

Goodnight et al. [33] ont exploré le potentiel de la programmabilité des GPU pour ajouter une reproduction de tons en temps réel à des applications graphiques interactives. Ils ont implémenté les versions globale et locale de l'algorithme photographique de Reinhard et al. Pour des images de 256×256 pixels, les taux de trames obtenus sont plus grands que 20 fps. Cependant, pour des images de 512×512 pixels, le système n'atteint que 5 fps. Les auteurs ont observé que le goulot d'étranglement de l'opérateur global était le calcul de la moyenne logarithmique (environ 60%), et que pour l'algorithme local c'était le calcul des convolutions de la pyramide gaussienne.

D'un autre côté, Hassan [31] a implémenté trois approximations pour l'opérateur local de Reinhard sur une plateforme contenant un FPGA. Les fréquences d'opérations obtenues varient entre 65 et 93 MHz. Ces fréquences permettent un traitement vidéo en temps réel à 60 fps pour des images de 1024×768 pixels. Dans le même contexte, Vytla et al. [25] ont proposé des modifications à l'algorithme de Fattal. Ils ont atteint environ 100 fps pour une image de 1 Mpixel, à une fréquence d'horloge de 114 MHz. Cependant, il faut noter que ces résultats n'incluent ni le calcul du logarithme des luminances ni le calcul de la pyramide gaussienne.

Nous avons trouvé un seul travail de recherche lié aux processeurs sur mesure pour des applications de TM. Chiu et al. [5] ont développé un processeur de reproduction de tons basé sur un cœur d'un ARM avec un circuit intégré à application spécifique (ASIC). Leur processeur inclut une reproduction de tons photographique globale modifiée et une compression dans le domaine des gradients utilisant des blocs, basées sur les algorithmes proposés par Reinhard et al. [3] et Fattal et al. [2], respectivement. De plus, ils ont réalisé un profilage logiciel pour trouver les fonctions importantes en termes du nombre de cycles. Le processeur peut tourner à une fréquence d'horloge de 100 MHz et entraîne une amélioration de 50% en performance et surface par rapport à une de leurs implémentations antérieures. Cependant, cette approche n'offre pas la flexibilité souhaitée car quelques modules critiques sont implémentés sur un ASIC.

Quant aux applications connexes, Saponara et al. [7] ont proposé des processeurs à usage spécifique (ASIP) partiellement reconfigurables pour l'amélioration du contraste avec le filtrage Retinex. Ils ont implémenté cette technique sur une technologie CMOS de 180 nm et ont obtenu une performance comparable aux ASIC. De plus, le système possède une flexibilité proche des processeurs de signal numérique (DSP) à consommation de puissance plus faible.

1.2 Amélioration du contraste d'images LDR

Parmi les nombreuses applications de la reproduction de tons, l'amélioration du contraste joue un rôle clé dans le domaine du traitement d'images et de la vision artificielle. Il faut préciser que les techniques d'amélioration du contraste et les méthodes de reproduction de tons ont des buts et des propriétés différents [18]. La reproduction de tons part des valeurs de luminance du monde réel qui se distribuent dans une plage dynamique potentiellement grande, ce qui diffère des images LDR à améliorer. Ces luminances réelles ne contiennent pas de distorsions en tant que telles, au contraire des images en quête d'amélioration. De plus, l'un des objectifs de la reproduction de tons est d'obtenir une correspondance entre l'image résultante et la scène réelle en simulant la visibilité et le contraste. Par contre, les techniques d'amélioration d'images visent à augmenter autant que possible la visibilité et le contraste, sans se soucier de la correspondance [18]. Cependant, même si les algorithmes de reproduction de tons sont conçus pour comprimer la plage dynamique des images HDR, ils peuvent aussi être utiles pour ajuster le contraste et faire ressortir des détails d'une image classique à basse gamme dynamique [2].

En général, les techniques d'amélioration d'images LDR existantes peuvent être classifiées en deux catégories : les méthodes agissant dans le domaine spatial et les méthodes agissant dans le domaine des fréquences. Les méthodes spatiales fonctionnent sur les pixels de l'image, tandis que les méthodes basées sur les fréquences modifient les composantes spectrales de l'image [34]. Les méthodes spatiales peuvent être divisées en trois types : locales, globales et hybrides, tout comme les algorithmes de TM. Les méthodes locales considèrent une petite région de pixels voisins pour transformer chaque pixel [35]. Les méthodes globales tiennent compte de toute l'image, et les méthodes hybrides sont une combinaison des deux.

L'une des approches les plus populaires pour améliorer le contraste est l'égalisation d'histogramme. Arici et al. [35] ont proposé une méthode globale pour modifier un histogramme en résolvant un problème d'optimisation à deux critères qui minimise une fonction de coût. Le nouvel histogramme est une moyenne pondérée entre l'histogramme d'entrée et l'histogramme uniforme souhaité. Cette méthode inclut des critères additionnels tels que la robustesse au bruit et l'étirement blanc/noir, et gère les pointes des histogrammes. Kao et al. [8] ont adapté une méthode d'amélioration du contraste local (LCE). Cette méthode diffère d'une LCE normale du fait de l'utilisation du domaine logarithmique suivi par une normalisation afin d'enlever les valeurs négatives résultantes. Les inconvénients des méthodes basées sur des histogrammes incluent l'ajustement excessif du contraste, les halos non désirés et la faible amélioration des textures fines de l'image [8, 35].

Parmi d'autres techniques du domaine spatial, Panetta et al. [36] ont introduit une approche basée sur des caractéristiques du système visuel humain et l'égalisation à histogrammes multiples. C'est une méthode locale combinée avec quelques seuils globaux obtenus de l'image. Ils ont aussi proposé un autre algorithme local pour mieux préserver les détails aux arêtes tout en améliorant le contraste des images ayant un éclairage variable. Plus récemment, Chen et al. [34] ont donné une solution pour réduire quelques défaillances de l'amélioration d'images en fusionnant une image résultante améliorée et une image originale, basée sur le contraste local. Les défaillances ciblées étaient la perte de détails, la perte du contraste local et un « grisonnement » de l'image, c'est-à-dire que l'image tend à devenir grise.

En ce qui concerne les approches basées sur les fréquences, l'algorithme présenté par Lee [37] utilise la Transformée en Cosinus Discrète (DCT) pour comprimer la plage dynamique et pour

améliorer le contraste des images. Afin de comprimer la plage dynamique, les coefficients de basse fréquence sont modifiés en tenant compte de la théorie Retinex. Pour améliorer le contraste des images, les coefficients AC sont manipulés en utilisant une nouvelle mesure du contenu spectral de l'image.

1.3 Métriques de qualité des images

L'évaluation de la qualité des images est une tâche très complexe. Elle comprend des connaissances sur la physiologie de l'œil humain, les phénomènes psychophysiques en perception visuelle et l'objectif visé du traitement. En fait, la détermination d'une qualité élevée ou faible de l'image repose bien souvent sur la subjectivité et le problème à résoudre. Malheureusement, il n'existe pas de métrique objective unique qui donne des résultats significatifs pour toutes les images [35]. Autrement dit, il n'existe aucun moyen quantitatif universel pour spécifier la validité à la fois objective et subjective d'une méthode d'amélioration d'images [36]. Cependant, quelques mesures quantitatives peuvent indiquer un certain degré d'amélioration de la qualité.

Wang et Bovik [38] ont proposé de classer les différentes métriques d'après trois critères qui peuvent se chevaucher : connaissance de l'image originale, connaissance du processus de distorsion de l'image, et connaissance du système visuel humain. La première catégorie divise les métriques selon la disponibilité d'une image de référence de qualité « parfaite », à savoir : référence disponible, référence aveugle ou non disponible, et référence réduite si nous ne pouvons extraire que certaines propriétés statistiques de l'image de référence. Dans la deuxième catégorie, les métriques sont classées selon l'usage : générales ou spécifiques. Les méthodes d'évaluation à usage général sont conçues pour couvrir une grande plage d'applications. Les métriques spécifiques tiennent compte seulement d'un certain type de distorsion comme, par exemple, les effets de blocs issus d'une compression DCT par blocs. Finalement, la troisième catégorie regroupe les métriques selon le développement des méthodes. Elles peuvent simuler chaque composante du système visuel et chaque caractéristique psychophysique (approche ascendante), ou utiliser plutôt des hypothèses générales du système visuel (approche descendante).

1.3.1 Métriques pour images LDR

Les métriques les plus répandues requièrent qu'une image de référence soit disponible. Pour les images LDR, ces métriques s'inspirent en grande partie de l'erreur carrée moyenne (MSE, *Mean Squared Error*) [38], calculée entre une image de référence et l'image traitée. Pour une image de $m \times n$ pixels, le MSE est calculé comme suit :

$$MSE = \frac{1}{m \times n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \left(I_{référence} - I_{résultante} \right)^2. \tag{1.7}$$

Ainsi, le PSNR (*Peak Signal to Noise Ratio*) est défini à partir du MSE comme l'indique (1.8).

$$PSNR = 10 \log_{10} \left(\frac{L^2}{MSE} \right), \tag{1.8}$$

où L est le nombre d'intensités possibles de l'image. Pour les images de 8 bits/pixel, L = 255. Une image est considérée d'une qualité très élevée si sa valeur de PSNR est plus grande que 50 dB [4].

Parmi d'autres métriques de référence disponible, Wang et al. [39] ont développé une approche descendante : l'indice de similarité structurale (SSIM). Ils partent de l'hypothèse que le système visuel humain s'adapte facilement à l'extraction des propriétés qui représentent la structure des objets d'une scène, indépendamment de l'éclairement lumineux. Donc, leur métrique combine une comparaison de la luminance locale, du contraste local et de la structure entre les deux images. L'expression simplifiée de l'indice SSIM est donnée par (1.9) et est appliquée à une fenêtre 8×8 :

$$SSIM(\boldsymbol{x}, \boldsymbol{y}) = \frac{(2\mu_x \mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}, \tag{1.9}$$

où $\boldsymbol{x}$ et $\boldsymbol{y}$ représentent les deux images à comparer, μ est la moyenne d'intensités de l'image indiquée par le sous-indice, σ est l'écart-type de l'image aussi indiquée par le sous-indice, C_1 et C_2 sont des constantes pour éviter des instabilités, et σ_{xy} est le coefficient de corrélation entre $\boldsymbol{x}$ et $\boldsymbol{y}$ utilisé pour la comparaison de la structure. Pour évaluer la qualité globale d'une image, ils ont proposé la moyenne de l'indice SSIM, le MSSIM. Ses valeurs varient entre 0 et 1, où 1 indique que l'information structurale de l'image cible n'a pas été dégradée par rapport à l'image de référence.

En vue de mesurer l'amélioration du contraste dans des images LDR, certains auteurs ont utilisé l'erreur moyenne absolue de la brillance (AMBE) [40] et le contraste d'intensités [34]. AMBE calcule la différence absolue entre l'intensité moyenne d'entrée ($\bar{X}$) et celle de sortie ($\bar{Y}$), comme suit :

$$AMBE = |\bar{X} - \bar{Y}|. \tag{1.10}$$

Le contraste d'intensités C mesure la différence d'intensités entre une valeur de pixel et son voisin du côté droit sur toute l'image, basé sur la matrice normalisée de co-occurrences en niveau de gris g obtenue de l'image, tel qu'indiqué à (1.11).

$$C = \sum_{i,j} |i - j|^2 g(i,j), \tag{1.11}$$

où $i, j = 0,\ldots, L$ et L est le nombre de niveaux de gris [34].

D'autres auteurs ont utilisé l'entropie pour mesurer le contenu d'une image, car sa définition est liée à la variabilité [30, 35]. Nous supposons que si une image possède plus d'information visuelle, cela implique une amélioration du contraste [9]. Le calcul de l'entropie est comme suit :

$$ent(z) = -\sum_{i=0}^{L-1} p(z_i) \log_2 p(z_i), \tag{1.12}$$

où p représente l'histogramme de l'image z.

1.3.2 Métriques pour images HDR

Dans le cadre de l'imagerie HDR, nous sommes intéressés à évaluer les résultats de la reproduction de tons en supposant que l'image HDR est déjà une représentation fidèle de la scène en question. Tout d'abord, il faut savoir ce qui sera mesuré et quel est l'objectif de la reproduction pour faciliter l'évaluation. Čadík et al. [19] ont proposé plusieurs attributs à mesurer : la brillance, le contraste, la reproduction des couleurs, la reproduction des détails notamment dans des régions d'intérêt, les artéfacts et d'autres propriétés spéciales comme l'acuité visuelle. Ces attributs peuvent interagir entre eux. Par exemple, la perception du contraste est influencée par la reproduction des détails, la saturation des couleurs, la luminosité de l'image et si les contours sont bien marqués. De plus, les auteurs ont affirmé que les méthodes de TM mélangent trois approches de reproduction : approche perceptuelle, si l'intérêt repose sur la simulation du système visuel humain; approche cognitive, si il est important de comprendre plusieurs détails ou la structure du contenu de l'image; et approche esthétique, si ce qui importe

est l'apparence agréable de l'image. Nous nous concentrerons sur l'évaluation du contraste et de la reproduction des détails d'un point de vue perceptuel.

Nous avons trouvé quelques études subjectives à ce sujet. Čadík et al. [19] ont obtenu une métrique générale avec la plupart d'attributs mentionnés précédemment, à partir d'expériences psychophysiques incluant 14 algorithmes de TM, 3 scènes et 20 personnes. Leurs résultats montrent que la meilleure qualité a été obtenue en général par les algorithmes globaux. Ceci contredit la préférence généralisée des algorithmes locaux en termes de la qualité des images. De plus, ils ont conclu qu'il n'est pas obligatoire de fournir une image de référence pour évaluer la qualité des images résultantes.

D'autres évaluations subjectives ont été mises en œuvre par Ledda et al. [41] et Ashikhmin et al. [42]. L'approche de Ledda et al. est particulière, puisqu'ils ont utilisé un écran HDR au lieu des scènes réelles comme dans la plupart d'études. De plus, leur méthodologie diffère du classement traditionnel, car ils ont effectué les comparaisons entre une paire d'images résultantes des algorithmes de reproduction de tons et l'image HDR de référence. Ils ont évalué principalement la reproduction des détails de 6 algorithmes de TM, avec 23 scènes différentes et 48 personnes. Les auteurs ont remarqué que la couleur pourrait devenir un facteur important, puisque les résultats ont été différents lorsqu'ils ont utilisé des images en niveaux de gris. D'un autre côté, ils ont confirmé que les algorithmes locaux ont tendance à perdre du contraste pour laisser place à une meilleure reproduction des détails. Par ailleurs, Ashikhmin et al. [42] ont mesuré la qualité de 5 algorithmes, avec 4 ou 5 scènes dépendamment de l'expérience, et 15 personnes. Contrairement à Čadík et al., ils ont mis l'accent sur l'importance d'avoir l'image de référence HDR pour procéder à la comparaison. Leurs résultats les mènent à penser que les personnes préfèrent les images d'un contraste plus élevé que dans la réalité lorsqu'elles ne possèdent pas d'images de référence. Même si les analyses subjectives consultées ici ont été faites avec différents algorithmes, méthodologies, scènes et personnes, la conclusion semble être qu'il n'existe pas d'algorithme unique capable d'obtenir la meilleure qualité de l'image pour tous les types de scènes. En outre, c'est un sujet qui mérite encore plus de recherche à cause de manques de cohérence entre les diverses études.

En ce qui a trait aux métriques objectives dans le contexte de la reproduction de tons, une option est de convertir l'image LDR résultante en HDR et comparer ainsi deux images HDR.

Cependant, ce processus complexifie la tâche et entraîne plus de facteurs d'erreur. De plus, les métriques utilisées pour les images classiques requièrent des modifications afin d'obtenir des résultats cohérents avec la correspondance entre les images HDR et les scènes réelles. Pour ce faire, les valeurs de luminance des images HDR devraient être converties à un espace uniforme d'un point de vue perceptuel et qui soit adéquat pour la plage dynamique réelle des pixels de l'image. L'une des approches est l'espace des seuils différentiels (JND, *Just Noticeable Differences*) mis à l'échelle, mais c'est un sujet encore ouvert à la recherche [1].

Une autre option est de comparer directement une image HDR et sa version LDR respective. À cet effet, il est nécessaire de prendre en considération la différence entre les plages dynamiques concernées [43]. Aydin et al. [43] ont présenté une métrique indépendante de la plage dynamique (DRI, *Dynamic Range-Independent*) qui mesure des changements structurels par rapport à l'image de référence. La métrique génère un plan de distorsions à partir des pertes du contraste visible, de l'amplification du contraste invisible et du renversement du contraste visible associé aux artéfacts. Pour ce faire, ils ont inclus un modèle du système visuel humain qui simule l'optique de l'œil, la réponse non-linéaire des neurones de la rétine et la sensibilité visuelle au contraste en fonction des fréquences spatiales, basé sur le prédicteur de différences visibles HDR proposé par Mantiuk et al. [44]. D'autres auteurs [31] se contentent de choisir un algorithme réputé donner des images de bonne qualité et d'utiliser ses résultats comme images de référence. De cette façon, les métriques classiques peuvent être utilisées.

1.4 Conception de processeurs spécialisés

1.4.1 Vue d'ensemble

Pendant la dernière décennie, les besoins des systèmes embarqués portables sont devenus plus exigeants. Bien que la complexité des applications multimédia et mobiles ait augmenté, on s'attend à ce que les plateformes de développement soient plus flexibles, performantes, efficaces au niveau énergétique, moins dispendieuses et faciles à concevoir pour minimiser le temps de mise sur le marché autant que possible. Offrant un meilleur compromis entre la performance, la consommation de puissance et la flexibilité, les processeurs à jeu d'instructions spécialisées (ASIP, *Application-Specific Instruction-set Processor*) deviennent une alternative d'implémentation attirante [6, 7]. Les ASIP sont des processeurs conçus pour une application ou

un domaine spécifique. Ils permettent, par exemple, d'activer certaines caractéristiques ou des unités fonctionnelles, d'ajouter des instructions sur mesure, et de mettre à jour, de modifier ou de réutiliser le code de l'application sans avoir besoin d'une conception complètement nouvelle comme avec un ASIC [6].

Une méthodologie générique de conception d'ASIP consiste principalement en deux phases : une analyse de l'application et de la pré-architecture, et un raffinement de l'architecture et son implémentation, comme indiqué à la Figure 1.6. L'application est généralement codée en langage C. Tout d'abord, le but est de trouver les parties du code les plus fréquemment exécutées et qui requièrent un effort gourmand en calculs. Cette analyse aide à l'identification de quelques caractéristiques de la microarchitecture, à savoir : les unités fonctionnelles pertinentes, les types de données, la hiérarchie des mémoires, etc. Une autre étape qui en tire profit, c'est la définition de l'architecture de jeu d'instructions (ISA, *Instruction-Set Architecture*) de base en se passant des opérations rares et en déterminant des accélérateurs matériels à ajouter s'il y a lieu. Le produit de la première phase est une architecture initiale qui va être améliorée de manière itérative à la deuxième phase jusqu'à ce que les contraintes du design soient satisfaites. Il est donc nécessaire de vérifier la fonctionnalité à chaque modification et d'évaluer la performance et la surface, entre autres aspects, pour finalement obtenir le processeur spécialisé souhaité [6].

Récemment, nous [45] avons présenté une méthodologie qui combine l'ajout d'instructions spécialisées avec la réduction de l'ISA pour à la fois augmenter la performance et diminuer les coûts matériels. Le processeur proposé atteint un facteur d'accélération de 1,57× et une réduction de 45% en surface par rapport à un MicroBlaze de Xilinx. Les tests ont été faits pour une convolution 2D utilisant un noyau gaussien 5×5. Aubertin et al. [46] ont proposé une méthodologie qui utilise de manière efficace la bande passante mémoire disponible en exploitant le parallélisme de l'application cible. Ils ont proposé d'ajouter des instructions SIMD (*Single-Instruction Multiple-Data*) et des registres spécialisés ainsi que d'utiliser une architecture VLIW (*Very Long Instruction Word*), de façon à imiter un réseau systolique pipeliné. Ils ont obtenu des facteurs d'accélération d'entre 95× et 1330× pour des algorithmes de désentrelacement vidéo intra-trames, et d'entre 36× et 80× pour des convolutions 2D avec un processeur Xtensa LX2.

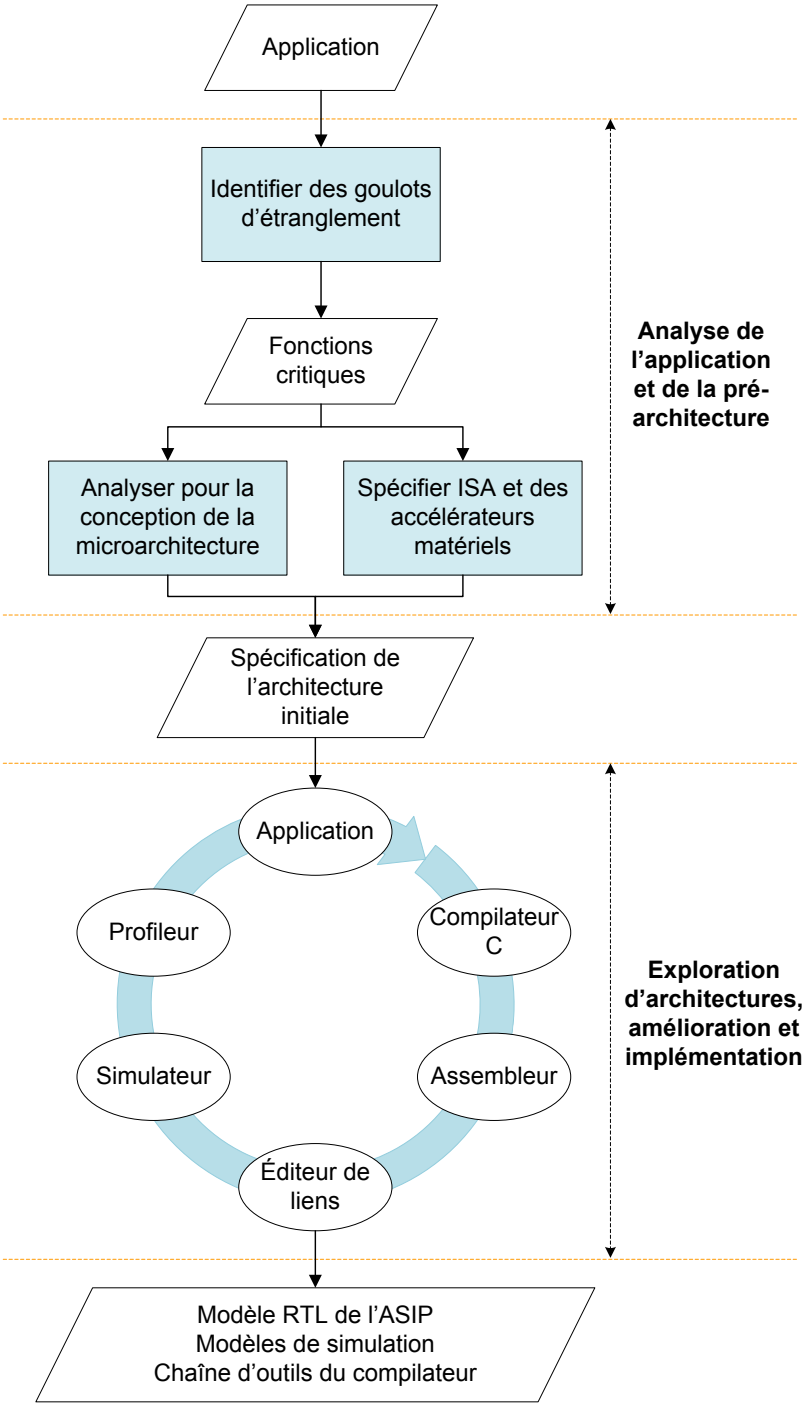


Figure 1.6: Étapes génériques pour la conception d'un ASIP [6]

En général, nous retrouvons deux types de philosophies pour la conception automatisée de processeurs spécialisés [6]. La première permet de développer des architectures en partant de zéro, ce qui favorise des niveaux élevés de particularisation. Dans cette tendance, les langages de

description architecturale (ADL, *Architecture Description Language*) facilitent la modélisation des processeurs. La deuxième philosophie accélère le processus de développement et de vérification via des cœurs de processeurs configurables et extensibles déjà préconçus. Dans tous les cas, il faut aussi considérer la génération d'outils de design, tels que le compilateur, le simulateur, l'assembleur et le débogueur. Nous décrirons ci-après ces deux approches en mettant l'accent sur le langage LISA (*Language for Instruction-Set Architecture*) et les outils de la compagnie Tensilica.

1.4.2 Langages de description architecturale

Les ADL sont attrayants pour les développements d'ASIP, principalement en raison de leur processus de conception simple par rapport aux langages de description matérielle (HDL, *Hardware Description Languages*). D'un point de vue matériel, les ADL capturent la structure et/ou le comportement d'architectures des processeurs. La structure représente la vue du concepteur matériel, tandis que le comportement porte sur la vue du programmeur de l'architecture [47]. Ces langages permettent donc de saisir les caractéristiques propres du matériel, comme le parallélisme et la synchronisation, à la différence des langages de programmation. De plus, ils possèdent un niveau d'abstraction adéquat pour l'analyse et l'exploration d'architectures. Ceci devient compliqué avec les HDL, dont le niveau d'abstraction est trop bas pour extraire le jeu d'instructions de l'architecture [48].

Les ADL peuvent être classifiés par l'objectif et par le contenu. L'objectif correspond au produit principal obtenu de la description par ADL. Il en existe quatre catégories : orientées vers la simulation, la synthèse, la compilation et la validation. En ce qui concerne le contenu, il indique la nature de l'information capturée par le langage. Nous y retrouvons des langages structurels, comme MIMOLA, qui sont axés sur la description des blocs fonctionnels et de leurs interconnexions. Il existe aussi des langages comportementaux, tels que nML et ISDL, utilisés pour décrire le jeu d'instructions et le comportement du processeur. Une troisième classe regroupe les ADL mixtes, capables de capturer à la fois la structure et le comportement, tels que LISA et EXPRESSION [47, 48].

Un modèle LISA comporte deux types d'éléments : ressources et opérations. Les ressources décrivent les unités d'entreposage et du chemin de données, telles que la mémoire, le fichier de

registres, le pipeline, les interconnexions globales et les unités fonctionnelles. Les opérations représentent les instructions, principalement à l'aide de trois sections [48, 49] :

- « CODING » contient la représentation binaire requise pour le décodeur d'instructions;
- « SYNTAXE » décrit une chaîne de caractères spécifiant les identificateurs de l'instruction et les opérandes à utiliser dans le langage assembleur;
- « BEHAVIOR » correspond à l'exécution de l'instruction, semblable au corps d'une fonction décrite avec le langage de programmation C.

Afin d'incrémenter l'efficacité de la conception, il est possible d'exploiter les caractéristiques communes d'instructions similaires via une hiérarchie d'opérations et la déclaration d'opérandes génériques. De plus, dans les modèles précis au cycle d'horloge, le langage permet d'associer chaque opération à l'étage respectif du pipeline. D'ailleurs, la section appelée « ACTIVATION » est utile pour planifier l'exécution d'opérations subordonnées dans des étages postérieurs du pipeline [48, 49].

Le langage LISA est utilisé au sein des outils du logiciel *Processor Designer* de la compagnie Synopsys. À partir du modèle LISA du processeur, le logiciel peut générer un compilateur C, un assembleur, un éditeur de liens et un simulateur. Ces outils aident à mesurer le nombre de cycles d'horloge ou d'instructions et à identifier les goulots d'étranglement de l'application cible. Une fois que le modèle LISA est satisfaisant, il est possible de générer de manière automatique la description HDL (synthétisable) en vue de mesurer la fréquence d'horloge, la surface du processeur et la consommation de puissance. Cependant, la description matérielle résultante n'est pas en général aussi efficace qu'un modèle développé à la main par un concepteur expérimenté. C'est pourquoi ces descriptions RTL sont utilisées plutôt dans la phase d'exploration d'architectures pour obtenir une estimation des caractéristiques physiques du processeur. Pour l'implémentation finale de l'architecture, des améliorations manuelles sont souvent requises [47].

Un des modèles LISA fourni avec l'outil *Synopsys Processor Designer* est celui du processeur LTRISC. Ce processeur RISC de 32 bits a une architecture Harvard avec 16 registres à usage général et du pipeline à quatre étages. Le jeu d'instructions supporte des opérations arithmétiques entières de base, des instructions charge-sauvegarde à usage général, des instructions de

comparaisons et des branchements. De plus, le modèle de ce processeur permet d'effectuer des simulations précises au cycle d'horloge près et de générer la description RTL correspondante.

1.4.3 Processeurs configurables préconçus

Plusieurs compagnies offrent des processeurs préconçus et pré-vérifiés pouvant être paramétrés et étendus. C'est le cas des processeurs Xtensa de la compagnie Tensilica[1], Nios II d'Altera[2], MicroBlaze de Xilinx[3], LEON d'Aeroflex Gaisler[4] et les cœurs ARC de Synopsys[5]. Le processeur Xtensa est l'un des processeurs configurables les plus populaires [6]. C'est un processeur de type RISC à 32 bits, d'architecture Harvard et à jeu d'instructions registre-registre. Il comporte un total de 32 ou 64 registres à usage général et utilise du fenêtrage pour la gestion de paramètres lors d'un appel de fonction. Ses instructions sont encodées sur 24 bits et certaines peuvent être sur 16 bits. Son pipeline est à 5 étages et peut être étendu à 7 [50].

Le processeur Xtensa LX2 est un processeur paramétrable, étant donné qu'il offre la possibilité d'activer et de modifier certaines caractéristiques prédéterminées comme la taille de mémoires cache, des multiplicateurs, des unités à virgule flottante, le type de boutisme, le nombre de niveaux du pipeline, des instructions pour accélération de boucles, etc. C'est aussi un processeur extensible, car il est possible de rajouter des instructions spécialisées, des unités fonctionnelles, des registres de contrôle et d'autres éléments qui n'avaient pas été considérées par les concepteurs originaux [50].

L'extension des fonctionnalités mentionnées précédemment se fait à l'aide du langage TIE (*Tensilica Instruction Extension*), similaire à Verilog. Le *TIE Compiler* les intègre au processeur de base en générant une chaîne d'outils du compilateur, un modèle de simulation et du matériel au niveau de registres (RTL, *Register Transfer Level*) pour le cœur modifié [6]. Les instructions sur mesure sont décrites au sein d'une « opération » TIE et, une fois compilées, elles peuvent être utilisées dans le code cible C/C++ ou en assembleur comme n'importe quelle instruction. Il est possible de combiner multiples opérations en une seule instruction via une fusion d'opérations ou

---

[1] Tensilica's Xtensa Customizable Processors (http://www.tensilica.com/products/xtensa-customizable.htm)
[2] Altera's Nios II Processor (http://www.altera.com/devices/processor/nios2/ni2-index.html)
[3] Xilinx's MicroBlaze Soft Processor Core (http://www.xilinx.com/tools/microblaze.htm)
[4] Aeroflex Gaisler (http://www.gaisler.com)
[5] Synopsys' ARC Processor Cores (http://www.synopsys.com/IP/ProcessorIP/ARCProcessors)

une vectorisation SIMD. De plus, l'Xtensa LX2 offre la technologie FLIX (*Flexible-Length Instruction Extensions*), la version de Tensilica d'une architecture VLIW. Cela permet de concevoir des mots d'instruction de 32 ou 64 bits et ainsi d'exécuter différentes instructions simultanément [50].

Tensilica fournit aussi un outil permettant une particularisation complètement automatisée. Le compilateur XPRES suggère de différentes configurations les plus appropriées pour l'application cible. De plus, XPRES peut générer des descriptions TIE pour le processeur sélectionné [6, 50].

CHAPITRE 2 CONCEPTION D'UN PROCESSEUR SPÉCIALISÉ POUR UN ALGORITHME GLOBAL DE REPRODUCTION DE TONS

Les algorithmes de reproduction de tons peuvent être utiles pour ajuster le contraste et faire ressortir des détails d'une image classique à basse gamme dynamique (LDR), même s'ils sont conçus pour comprimer la plage dynamique des images HDR [2]. Nous avons mis en œuvre une analyse comparative de quatre méthodes d'amélioration du contraste trouvées dans la littérature. Parmi elles, deux méthodes de reproduction de tons ont été choisies. La section 2.1 porte sur ces comparaisons, et est basée sur l'article « *Comparative Analysis of Contrast Enhancement Algorithms in Surveillance Imaging* », publié dans le compte rendu de conférence de l'ISCAS 2011 [9].

Le reste du chapitre présente les points importants sur la conception de notre premier ASIP. Dans ce cas, nous avons utilisé le langage de description architecturale LISA (*Language for Instruction Set Architecture*), l'un des ADL les plus connus (voir plus de détails à la section 1.4.2). Nous avons choisi le processeur LTRISC comme le processeur de base, fourni avec l'outil *Synopsys Processor Designer*. Le processeur spécialisé proposé augmente la performance de la méthode globale proposée par Reinhard et al. [3], un des algorithmes utilisés aussi pour améliorer le contraste. Cet algorithme est connu grâce à sa faible complexité par rapport à d'autres méthodes de TM et à la haute qualité d'images résultantes dans le contexte de l'imagerie HDR [5]. De plus, il existe dans la littérature quelques implémentations matérielles qui serviront de base pour les comparaisons [5, 31, 33]. Les sections suivantes sont basées en grande partie sur l'article « *Customized Embedded Processor Design for Global Photographic Tone Mapping* », publié dans le compte rendu de conférence de l'ICECS 2011 [10].

Comme mentionné dans la revue de littérature (section 1.1.1), l'algorithme de Reinhard a été inspiré des techniques photographiques. D'abord, l'approche globale consiste à mettre à l'échelle les valeurs de la luminance en tenant compte de jusqu'à quel point l'impression globale de une scène est claire ou sombre. Ensuite, les valeurs élevées de la luminance sont comprimées via une transformation linéaire après avoir déterminé la valeur qui représente un blanc pur. Nous expliquerons les détails à la section 2.2. Par la suite, la section 2.3 contient une analyse des besoins en calculs et mémoire, suivie des instructions spécialisées proposées à la section 2.4.

Finalement, les opérations qui présentent de plus grands défis d'implémentation sont décrites à la section 2.5.

2.1 Analyse comparative d'algorithmes d'amélioration du contraste d'images LDR

Dans ce travail, nous avons comparé deux algorithmes d'amélioration du contraste et deux méthodes de TM utilisées avec le but d'améliorer le contraste d'images LDR, dans le contexte d'applications de surveillance. Pour ce type d'applications, l'ajustement du contraste en temps réel est une étape requise lorsque les vidéos sont prises dans des conditions de faible éclairage. L'approche du suivi visée repose principalement sur la détection d'arêtes. Si le suivi des objets est basé sur le modèle d'arrière-plan, l'amélioration du contraste pourrait créer de fausses détections quand elle est effectuée pour chacune des images de la vidéo cible. Dans ce cas, l'ajustement du contraste devrait être effectué à chaque fois que les changements de luminosité soient assez grands afin de mettre à jour le modèle d'arrière-plan.

Deux critères ont été considérés pour faire l'analyse comparative des quatre méthodes choisies : la qualité visuelle de l'image résultante et la performance (considérée comme l'effort de calcul). Les algorithmes comparés sont les suivants : la technique locale proposée par Fattal et al. [2], la méthode globale présentée par Arici et al. [35], la version globale de l'algorithme de Reinhard [3], et l'égalisation d'histogramme global (GHE). Les méthodes de Fattal et de Reinhard ont été prises de la bibliothèque de programmes PFStmo [11] qui contient 8 opérateurs de reproduction de tons de pointe codés en C++. L'algorithme de Fattal a été choisi pour représenter les techniques locales, et en raison de sa complexité modérée par rapport aux divers algorithmes locaux de TM (voir le Tableau 1.1). L'algorithme de Reinhard a été choisi étant donné sa popularité dans le domaine de l'imagerie HDR et sa faible complexité entre les méthodes de reproduction de tons. Ces deux méthodes seront expliquées dans les sections suivantes. L'algorithme d'Arici a été ré-implémenté à partir de la description de l'article original. Celui-ci a été choisi car il est récent et grâce à son accent sur l'efficacité de calcul. L'égalisation d'histogramme a été choisie du fait qu'elle est une des techniques les plus connues pour ajuster le contraste d'une image LDR.

Une méthodologie systématique a été suivie pour comparer les quatre approches choisies :

1. Sélection des images test

Deux types d'images ont été utilisés pour tester les algorithmes. Le premier groupe correspond à huit images représentatives de notre application cible. Dans ce cas, nous étions intéressés à surveiller le comportement d'un animal de laboratoire. Plus exactement, notre cible visait un rat dans une cage. Toutes ces images ont été prises en conditions de faible éclairage. Le deuxième groupe inclut deux images test souvent utilisées dans la littérature, prises d'une bibliothèque en ligne [13].

2. Ajustement des paramètres de chaque algorithme

Comme la qualité de l'image dépend souvent de la configuration des valeurs des paramètres [36], il a fallu les régler avec précision. D'abord, les valeurs ont été choisies selon les valeurs par défaut ou celles recommandées dans les articles originaux. Par la suite, les valeurs ont été ajustées après une évaluation subjective des images et les résultats de métriques objectives de qualité.

3. Test d'exploration

Une image de l'application cible avec un bon niveau de contraste a été traitée par les techniques sélectionnées. Une fois que des résultats satisfaisants ont été obtenus, le jeu de paramètres a été choisi et considéré acceptable pour l'évaluation de la méthode. Les valeurs des paramètres d'essai sont résumées au Tableau 2.1.

Tableau 2.1: Valeurs des paramètres pour tester les algorithmes d'amélioration du contraste

Algorithme	**Paramètres**	**Valeurs**
Fattal02	A	10
	β	0,9
	saturation	0,6
Arici09	λ	2
	A	0
	seuil	4
Reinhard02 - global	A	0,45
	Lwhite	1,5
	γ	1,3
GHE	Aucun	Aucune

4. Évaluation des méthodes

Nous avons considéré deux critères pour l'évaluation : l'amélioration de la qualité visuelle des images et la complexité mesurée comme le temps d'exécution. Outre l'évaluation subjective, nous avons utilisé plusieurs métriques pour l'amélioration du contraste. Il faut préciser que dans ce travail, la qualité de l'image fait allusion à l'ajustement du contraste global et dans quelques régions d'intérêt, notamment les zones sombres. Pour obtenir un indice de l'amélioration du contraste global, nous avons calculé l'erreur moyenne absolue de la brillance (AMBE) et le contraste d'intensités relatif à l'image originale. En ce qui concerne les régions ombragées, le contraste d'intensités ainsi que l'entropie ont aussi été calculés. Les définitions de ces métriques se trouvent à la section 1.3. La section 4.1 contient les résultats de ces comparaisons.

2.2 Description de l'algorithme de Reinhard

Une des méthodes d'amélioration du contraste évaluée a été celle de reproduction de tons proposée par Reinhard et al. [3]. Dans le cadre de l'imagerie HDR, sa version globale bénéficie d'une bonne réputation grâce à la qualité de ses images et aux faibles besoins en calculs par rapport à d'autres méthodes. Les modules principaux de cet opérateur sont montrés à la Figure 2.1 et décrits ci-après. Le cœur de l'algorithme est formé des deux blocs surlignés.

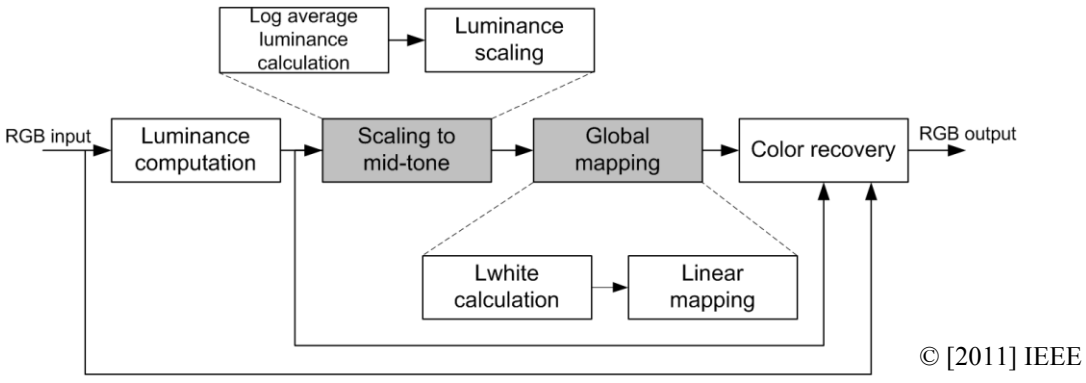


Figure 2.1: Diagramme de blocs de l'algorithme de reproduction de tons de Reinhard

L'image HDR est tout d'abord convertie de l'espace de couleur RGB vers des valeurs de luminance « renvoyées à la scène » (ou luminance réelle) comme suit :

$$L_w = 0.265R + 0.670G + 0.064B. \quad (2.1)$$

Par la suite, le bloc « *scaling to mid-tone* » est en charge de la première mise à l'échelle de la luminance. De manière similaire aux pratiques de photographie, une approximation de la clé de la scène est identifiée en calculant la valeur de la luminance logarithmique moyenne :

$$L_{avg} = exp\left(\frac{1}{N}\sum \log(\delta + L_w)\right), \tag{2.2}$$

où N est le nombre de pixels dans l'image et δ est une valeur petite (par exemple 0,0001) pour éviter les inconsistances quand la valeur de la luminance est zéro. Cette valeur indique jusqu'à quel point la vue d'ensemble de la scène est claire ou sombre [1].

Puis, la mise à l'échelle initiale est calculée avec (2.3), où la moyenne logarithmique est associée à une valeur désirée α. Pour des scènes à clé moyenne, α correspond à 18% de la plage d'affichage.

$$L = \frac{\alpha}{L_{avg}} L_w \tag{2.3}$$

Ensuite, la luminance compressée L_d est obtenue grâce à une assignation linéaire basée sur (2.4), comme suit :

$$L_d = \frac{L\left(1+\frac{L}{L_{white}^2}\right)}{1+L}, \tag{2.4}$$

où L_{white} est la valeur de luminance la plus petite à être transformée en blanc pur. Par défaut, L_{white} est la valeur maximale de la luminance après la mise à l'échelle initiale.

L'étape finale consiste à récupérer l'image couleur. Afin d'éviter des décalages couleur, le rapport entre la luminance compressée et la luminance réelle est maintenu constant [1]. Donc, l'image couleur résulte du calcul (2.5).

$$RGB_d = \frac{L_d}{L_w} RGB \tag{2.5}$$

2.3 Besoins en calculs et mémoire

Dans notre cas, le processeur devrait être capable de traiter au moins 25 images par seconde (fps). Pour évaluer les besoins en calculs et mémoire de l'algorithme de Reinhard, ses calculs dans les différentes étapes ont été considérés. Nous avons analysé tout l'algorithme à l'aide du graphe de

flot de calcul, montré à la Figure 2.2. Dans cette figure, le nombre d'exécutions de chaque opération pour une image de *N* pixels est noté à gauche.

À partir de la Figure 2.2 et les équations correspondantes présentées à la section 2.1, nous remarquons que le calcul de la moyenne de la luminance logarithmique (2.2) a besoin de la luminance réelle de chaque pixel (2.1). Cette étape doit être répétée pour chaque pixel dans chaque image pour calculer la sommation de (2.2). De plus, les valeurs L_w doivent être stockées dans la mémoire des données pour éviter de recalculer la luminance réelle. Tel qu'observé à la Figure 2.2, les valeurs de la luminance mise à l'échelle (L) et les valeurs de la luminance réelle (L_w) doivent être rechargées de la mémoire pour calculer (2.4) et (2.5), respectivement. En outre, les données RGB sont chargées deux fois, au début et à la fin, et doivent donc être sauvegardées dans la mémoire de données. En conséquence, il est nécessaire d'avoir 5 fois la taille de l'image, en mots de 32 bits, pour enregistrer R, G, B, L_w et L.

Afin d'estimer les besoins en calculs de l'algorithme global de Reinhard, nous avons tenu compte du nombre de fois que chaque opération doit être exécutée. Le Tableau 2.2 montre les résultats de cette estimation, tirée de la Figure 2.2. Les instructions de chargement, de sauvegarde et de contrôle sont exclues dans cette estimation.

Tableau 2.2: Opérations requises pour le calcul de chaque image

Opération	**Nombre de fois**
Multiplication	8×Taille de l'image
Division	4×Taille de l'image
Sommes/soustractions	6×Taille de l'image
Min/Max	1×Taille de l'image
Logarithme	1×Taille de l'image

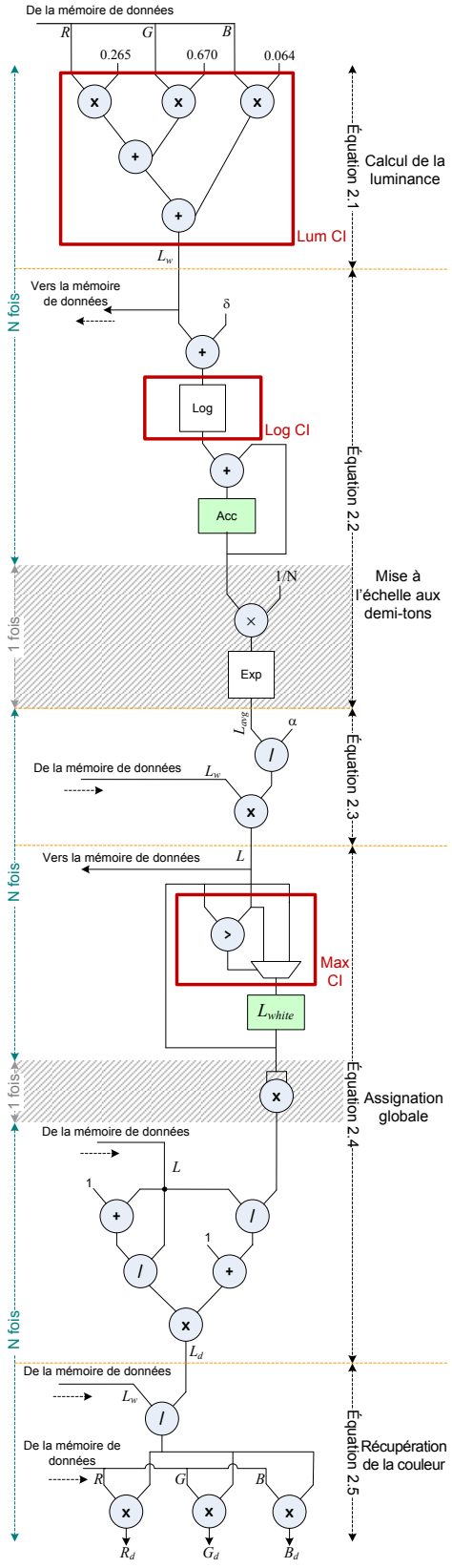


Figure 2.2: Flot de calcul de l'algorithme de Reinhard

2.4 Jeu d'instructions spécialisées proposé

Après avoir analysé la complexité des calculs de l'algorithme de Reinhard, nous l'avons analysé pour trouver des candidats adéquats visant l'accélération avec des instructions spécialisées (CI, *Custom Instructions*). À cet effet, le diagramme de flot de calcul de l'algorithme (Figure 2.2) a été analysé à nouveau et les parties les plus appropriées ont été extraites. Le processus de sélection et d'extraction a été réalisé manuellement en tenant compte des trois critères suivants :

- Les facteurs d'amélioration de la performance, y compris la fréquence d'utilisation et la réduction des délais des CI.
- Le coût matériel de la CI, y compris la surface et les effets probables sur la fréquence d'horloge dans l'ensemble.
- Les influences sur le processeur cible, y compris le nombre requis de ports d'entrée et de sortie dans le fichier des registres et les modifications à l'étape de décodage.

Considérant ces critères et le graphe de calculs, trois CI candidates ont été extraites : Lum CI, Log CI et Max CI, pour calculer la luminance, le logarithme et la valeur maximale de la luminance, respectivement. Ces instructions spécialisées sont soulignées à la Figure 2.2. À l'aide du langage de description architecturale LISA, nous avons ajouté ces trois instructions au processeur LTRISC, fourni par l'outil *Synopsys Processor Designer* et décrit à la section 1.4.2. Nous avons aussi ajouté des instructions de multiplication en virgule fixe et de division entière au jeu d'instructions original, mais celles-ci sont considérées comme faisant partie du processeur de base.

2.5 Logarithme et fonction exponentielle

Dans l'algorithme de Reinhard et al., une opération pour le calcul du logarithme est requise pour chaque pixel afin de calculer la moyenne des luminances dans le domaine logarithmique, tel qu'illustré à la Figure 2.2. La vitesse du calcul du logarithme a donc un impact important sur la performance de tout l'algorithme. Il est aussi nécessaire de calculer une fonction exponentielle pour retourner au domaine linéaire des luminances. La réalisation de ces deux opérateurs est l'un des défis principaux de l'implémentation matérielle des algorithmes de reproduction de tons. Afin

de simplifier l'implémentation, la luminance logarithmique moyenne et la fonction exponentielle sont calculées en base 2.

Nous avons développé une technique spécifique à faible coût, basée sur l'approximation de Mitchell pour calculer le logarithme [51]. De nombreuses techniques ont été proposées dans la littérature pour améliorer la précision de cette méthode [52], mais nous avons constaté qu'aucune ne satisfaisait bien nos besoins. Les valeurs de l'erreur introduites par la méthode de Mitchell suivent un patron similaire lorsque l'entrée varie entre 1 et 2, 2 et 4, 4 et 8, et ainsi de suite en base 2. La Figure 2.3 montre les valeurs de l'erreur pour le premier intervalle. Étant donné que (2.2) est une sommation de logarithmes et que les valeurs d'entrée sont distribuées de façon uniforme, nous pouvons améliorer la précision en ajoutant la valeur de l'erreur moyenne à la sortie de la méthode originale de Mitchell. Cette simple modification amène à une compensation suffisante de l'erreur dans le résultat de la sommation. Cette technique est développée en logiciel et Log CI implémente ses parties les plus complexes en matériel. La fonction exponentielle est approximée par la méthode de Mitchell corrigée avec une table de recherche (LUT) de 64 mots.

Contrairement à Chiu et al. [5] qui ont utilisé une longueur de mot uniforme pour toutes les variables intermédiaires, nous avons adopté une approche de longueur de mot multiple pour atteindre une efficacité plus élevée. Le processus de détermination de la longueur de mot a été effectué manuellement.

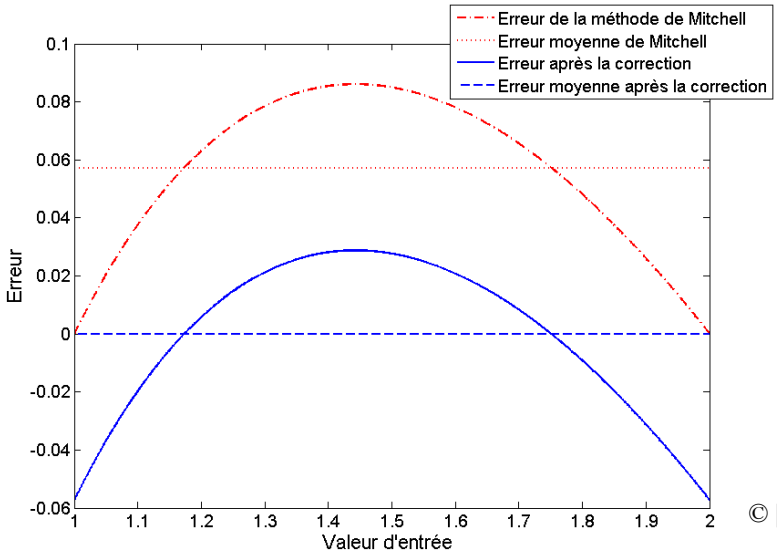


Figure 2.3: Erreurs de la méthode de Mitchell avant et après la correction

CHAPITRE 3 CONCEPTION D'UN PROCESSEUR SPÉCIALISÉ POUR UN ALGORITHME LOCAL DE REPRODUCTION DE TONS

Le présent chapitre contient une description de la conception de notre deuxième ASIP. Dans ce cas, nous avons utilisé le processeur configurable et extensible Xtensa LX2, dont les caractéristiques sont décrites à la section 1.4.3. Comme on l'a mentionné, les algorithmes globaux de reproduction de tons appliquent une fonction de correspondance à partir d'une vue d'ensemble de l'image. Un des désavantages est que cette fonction ne préserve pas toujours d'importants contrastes locaux. Par conséquent, nous avons visé un algorithme local de reproduction de tons dans cette occasion.

D'abord, nous avons choisi l'algorithme proposé par Fattal et al. [2]. Nous l'avons aussi utilisé dans l'analyse comparative d'algorithmes d'amélioration du contraste, décrit au chapitre précédent. Cette méthode possède une complexité modérée par rapport aux divers algorithmes locaux de TM (voir le Tableau 1.1). Cependant, son temps de calcul et ses besoins en mémoire sont très élevés. C'est pourquoi Vytla et al. [25] ont proposé une solution permettant de traiter les données par fenêtres au lieu d'avoir besoin de toute l'image. Le principe de base de cette approche est essentiellement le même de Fattal. Donc, nous avons décidé de l'accélérer à l'aide d'un processeur spécialisé.

Tel que discuté dans la revue de littérature, le principe de l'algorithme de Fattal et al. réside dans l'atténuation des magnitudes des gradients les plus grands à plusieurs échelles. Les gradients sont associés aux changements drastiques de la luminance. De cette manière, ils laissent intacts les détails et les textures correspondant à de faibles gradients, tout en réduisant la plage dynamique. Afin de reconstruire l'image à partir des gradients manipulés, il est nécessaire de résoudre l'équation de Poisson.

L'organisation du chapitre est comme suit. La section 3.1 présente une description générale de l'approche modifiée de Fattal et proposée par Vytla et al. Ses détails sont donnés à la section 3.2, suivis d'une estimation des besoins en calculs et mémoire à la section 3.3. Finalement, la section 0 décrit les instructions spécialisées proposées.

3.1 Description générale de l'algorithme de Fattal

Les modules de la méthode de reproduction de tons proposée par Fattal et al. [2] se trouvent à la Figure 3.1. Comme dans la plupart d'algorithmes de TM, cette méthode opère dans le domaine des luminances. Donc, une récupération de la couleur est requise à la fin des calculs. De plus, les étapes des normalisations sont suggérées par Fattal afin d'obtenir des atténuations qui affectent la même plage de valeurs pour diverses images [53]. Les blocs principaux sont surlignés et décrits ci-dessous.

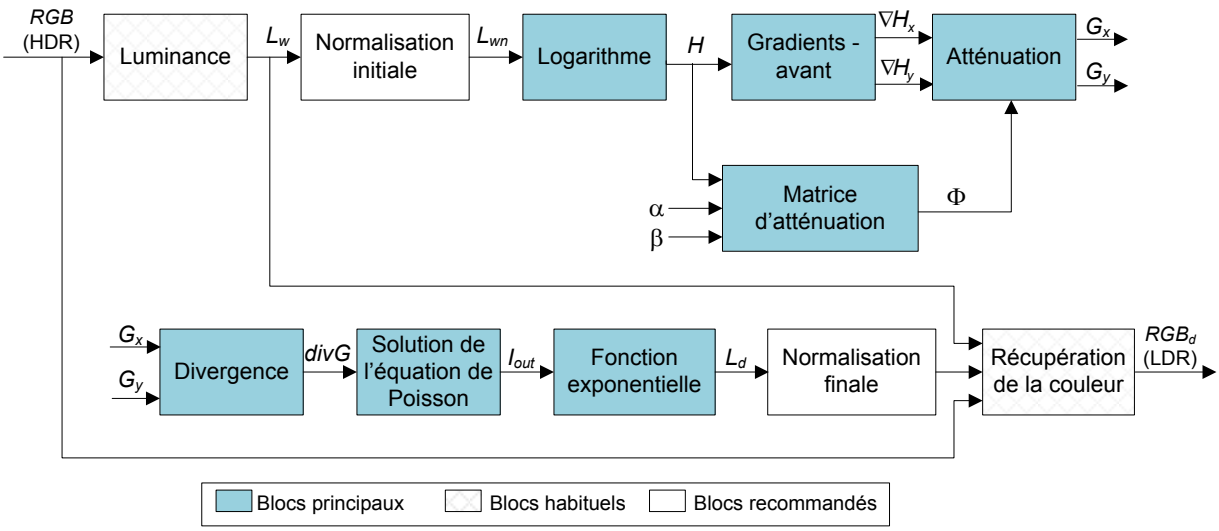


Figure 3.1: Blocs de l'algorithme de reproduction de tons de Fattal

Tout d'abord, le logarithme de la luminance est calculé, car il est considéré comme une approximation de la brillance aperçue. De plus, les gradients des valeurs logarithmiques correspondent à des contrastes locaux de la luminance. Ensuite, les gradients des valeurs du logarithme sont calculés, approximés par des différences-avant. Le bloc « Matrice d'atténuation » s'occupe de calculer les facteurs d'atténuation pour chaque pixel à partir de l'information des gradients par différences centrales obtenus à plusieurs échelles. Puis, le bloc « Atténuation » multiplie chaque facteur d'atténuation par la magnitude de chaque gradient-avant. À ce stage, les valeurs ont une plage réduite mais sont encore dans le domaine des gradients. Pour reconstruire l'image, la divergence des gradients atténués est calculée et l'équation de Poisson résultante est résolue. Ceci donne une image en valeurs logarithmiques. Finalement, la fonction inverse du logarithme est calculée pour obtenir les valeurs de la luminance [2, 25].

3.2 Description détaillée de la solution de Vytla

En général, le calcul de la solution de l'équation de Poisson est le goulot d'étranglement de l'algorithme de Fattal [5, 25]. De nombreuses méthodes existent pour résoudre cette équation. Les méthodes les plus souvent utilisées incluent des techniques multi-grilles [2] et la transformée de Fourier rapide [25]. D'autres auteurs proposent d'utiliser la transformée en sinus discrète (DST) par blocs [5].

Les approches traditionnelles existantes présentent une complexité élevée et ont besoin des conditions aux limites de l'image pour résoudre l'équation de Poisson. C'est pourquoi ces solutions requièrent l'image complète, ce qui n'est pas convenable en général lors d'une implémentation temps réel. Afin de pallier cette contrainte, Vytla et al. [25] ont proposé une solution directe locale en parcourant l'image avec des fenêtres 3×3 qui entourent chaque pixel. Cette solution n'utilise que des informations locales et a donc besoin de calculs différents pour la divergence et les facteurs d'atténuation. La Figure 3.2 montre les blocs qui présentent d'importantes modifications (les facteurs d'atténuation et la solution de l'équation de Poisson), ceux qui sont ajoutés et ceux qui sont légèrement modifiés. Ces différentes modifications sont expliquées dans les sections suivantes.

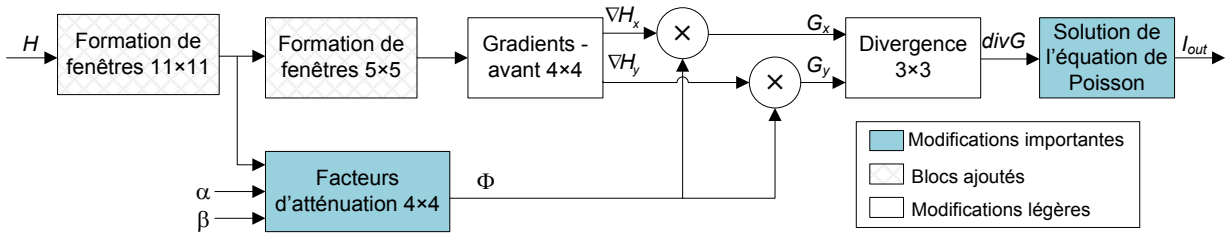


Figure 3.2: Modules principaux de l'algorithme modifié de Fattal utilisant des fenêtres

3.2.1 Calcul du logarithme

De la même façon que Fattal et al., l'approche de Vytla calcule le logarithme de la luminance. Afin de simplifier l'implémentation, le logarithme en base 2 est employé, comme montré à (3.1) :

$$H = \log_2(L_{wn} + \delta), \tag{3.1}$$

où L_{wn} est la luminance normalisée et δ est une valeur très petite (par exemple 0,0001).

3.2.2 Formation de fenêtres

Dans la solution proposée par Vytla et al., le module qui résout l'équation de Poisson a besoin d'une fenêtre de divergences 3×3 pour chaque pixel. Pour obtenir cette fenêtre, toutes les étapes précédentes, montrées à la Figure 3.2, doivent exécuter les calculs par fenêtres en tenant compte des conditions aux limites. L'utilisation de la condition de Dirichlet donne de meilleurs résultats en termes de la qualité de l'image et de la complexité de l'implémentation matérielle [25]. En conséquence, les valeurs autour des fenêtres 3×3 sont mises à zéro, ce qui fait étendre les fenêtres à 5×5 pixels. Cependant, dépendamment du calcul à effectuer, les fenêtres peuvent être de 4×4 ou de 6×6 pixels. La référence [25] donne plus d'informations à ce sujet. Par ailleurs, des fenêtres 11×11 sont nécessaires étant donné notre implémentation de la matrice d'atténuation expliquée ci-après.

3.2.3 Matrice d'atténuation locale

Selon l'algorithme original de Fattal, les valeurs de la matrice d'atténuation sont obtenues à partir d'une pyramide gaussienne (voir sa définition à la section 1.1.2). Vytla et al. proposent d'utiliser l'image originale et quatre niveaux additionnels sans sous-échantillonner. C'est-à-dire que les images de tous les niveaux ont la même résolution originale [25]. Dans ce travail, nous calculons la banque de filtres gaussiens de façon itérative. Le niveau courant est donc obtenu à partir du filtrage du niveau précédent, tel qu'illustré à la Figure 3.3. Cela implique que nous avons un seul noyau de convolution, au lieu des quatre noyaux gaussiens différents appliqués à l'image originale tel que décrit en [25] et [31]. Pour simplifier les calculs, le noyau 2D choisi est celui défini par (3.2) [30].

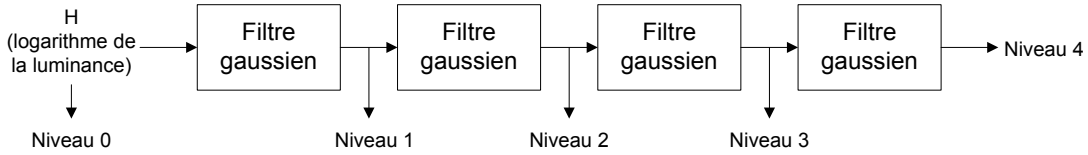


Figure 3.3: Structure du calcul de la pyramide gaussienne modifiée

$$w = \frac{1}{16} \cdot \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{3.2}$$

Pour chaque niveau *k* de la pyramide, il faut calculer une matrice 4×4 de gradients par différences centrales tel qu'indiqué en (3.3) :

$$\nabla H_k(x,y) = \left(\frac{I_k(x+1,y)-I_k(x-1,y)}{2}, \frac{I_k(x,y+1)-I_k(x,y-1)}{2}\right), \tag{3.3}$$

où I_k est H pour le niveau 0 et le résultat du filtre gaussien pour les niveaux postérieurs. Par la suite, les facteurs d'atténuation actuels sont obtenus à partir de (3.4) :

$$\varphi_k = \frac{\alpha \times Avg_k}{\|\nabla H_k\|}\left(\frac{\|\nabla H_k\|}{\alpha \times Avg_k}\right)^{\beta}, \tag{3.4}$$

ce qui peut se réécrire comme suit :

$$\varphi_k = \left(\frac{\|\nabla H_k\|}{\alpha \times Avg_k}\right)^{\beta-1}, \tag{3.5}$$

où α et β sont des paramètres choisis par l'utilisateur, $\|\nabla H_k\|$ est la norme du gradient central, et Avg_k est la moyenne des 16 normes des gradients centraux au niveau *k*. Ceci est une différence importante par rapport à l'algorithme de Fattal, car Fattal utilise la moyenne des normes de l'image entière. L'équation (3.5) est effectuée seulement si la valeur de la norme actuelle est plus grande que le produit $\alpha \times Avg_k$. Finalement, le calcul de la matrice d'atténuation définitive consiste à multiplier point à point les facteurs obtenus dans chaque niveau, comme suit :

$$\Phi = \prod_0^d \varphi_k, \tag{3.6}$$

où *d* est le nombre de niveaux de la pyramide sans compter l'image originale.

En somme, le calcul de la matrice d'atténuation pour chaque pixel, expliqué précédemment, se résume à la Figure 3.4. Il faut préciser que pour obtenir une fenêtre 3×3 au quatrième niveau de la pyramide, il est nécessaire d'avoir comme entrée une fenêtre de 5×5 pixels au troisième niveau, puisque le noyau de convolution utilisé contient 3×3 coefficients. De cette façon, la fenêtre 5×5 est obtenue à partir d'une fenêtre 7×7 et ainsi de suite.

3.2.4 Atténuation des gradients

Comme on l'a mentionné précédemment, le principe de l'algorithme de Fattal et al. réside dans l'atténuation des magnitudes des gradients les plus grands à plusieurs échelles, afin de réduire la plage dynamique de l'image. Vytla se base sur la même approche, sauf qu'une fenêtre 4×4 de gradients, au lieu d'une seule valeur, est calculée pour chaque pixel. Cette fenêtre est nécessaire

étant donné la solution de l'équation de Poisson, comme décrit à la section 3.2.2. Donc, pour chaque pixel, une fenêtre 4×4 de gradients est calculée par différences avant, comme suit :

$$\nabla H(x,y) \approx \big(H(x,y) - H(x+1,y), H(x,y) - H(x,y+1)\big). \tag{3.7}$$

Chaque gradient est ensuite atténué en multipliant la valeur de chacune des composantes avec le facteur d'atténuation Φ respectif, tel que décrit en (3.8) :

$$G(x,y) = \nabla H(x,y)\Phi(x,y). \tag{3.8}$$

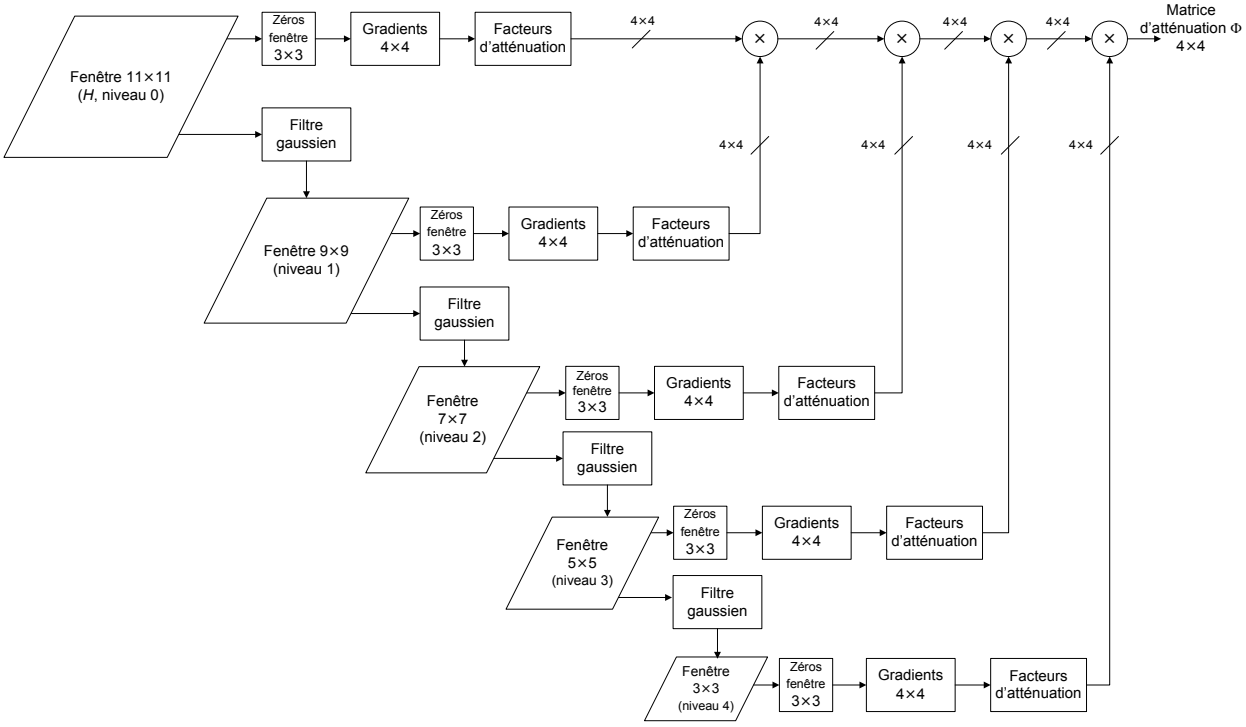


Figure 3.4: Blocs pour calculer la matrice d'atténuation locale

3.2.5 Reconstruction de l'image

L'algorithme de Fattal opère dans le domaine des gradients du logarithme de la luminance. Comme on l'a décrit ci-dessus, les magnitudes de ces gradients sont atténuées. Donc, il faut trouver l'image correspondante le mieux aux gradients modifiés. Pour ce faire, l'équation de Poisson donnée par (3.9) doit être résolue.

$$\nabla^2 I_{out} = divG, \tag{3.9}$$

où I_{out} est l'image cherchée en échelle logarithmique, $\nabla^2 I_{out}$ est l'opérateur laplacien et $divG$ est la divergence des gradients atténués. Au lieu de calculer une seule valeur de divergence par pixel comme Fattal, la solution de Vylta requiert une fenêtre 3×3 de divergences. Les valeurs des divergences sont estimées par différences arrière comme suit :

$$divG \approx \Big(G_x(x,y) - G_x(x-1,y) + G_y(x,y) - G_y(x,y-1)\Big), \tag{3.10}$$

où G_x et G_y sont la composante horizontale et la composante verticale des gradients modifiés, respectivement.

La solution directe de l'équation de Poisson conçue par Vytla et al. est donnée par (3.11) :

$$I_{out} = \frac{1}{16}\big(divG_{1,1} + divG_{3,1} + divG_{1,3} + divG_{3,3}\big) + \frac{1}{8}\big(divG_{2,1} + divG_{2,3} + divG_{1,2} + divG_{3,2}\big) + \frac{3}{8}divG_{2,2}, \tag{3.11}$$

où le premier indice de $divG$ correspond aux lignes et le deuxième aux colonnes à l'intérieur de la fenêtre 3×3 [25]. Finalement, comme I_{out} correspond aux valeurs du logarithme de la luminance, la luminance LDR s'obtient après avoir calculé la fonction inverse du logarithme, définie en (3.12) :

$$L_d = 2^{I_{out}} - \delta, \tag{3.12}$$

où δ est une valeur très petite ajoutée dans le calcul du logarithme (décrit à la section 3.2.1).

3.3 Besoins en calculs et mémoire

3.3.1 Estimation manuelle

Nous avons estimé les besoins en calculs et mémoire des modules principaux de la solution de Vytla, montrés à la Figure 3.1, avec notre calcul de la pyramide gaussienne modifiée. Pour ce faire, nous avons analysé l'algorithme à l'aide du graphe de flot de calcul de chaque étape. Les graphes suivants indiquent les opérations requises pour une image de *N* pixels. Chaque opération doit être exécutée le nombre de fois noté à gauche.

La Figure 3.5 présente le flot de calcul du logarithme de la luminance et de la formation des fenêtres. Dans ce cas, les détails d'implémentation du logarithme ne sont pas encore connus, raison pour laquelle nous le représentons par un bloc. Les valeurs du logarithme sont enregistrées

de façon à former des fenêtres 11×11 pour chaque pixel. Comme les instructions de chargement, de sauvegarde et de contrôle sont exclues dans cette estimation, seules une somme et une opération logarithme par pixel sont nécessaires dans ces deux étapes.

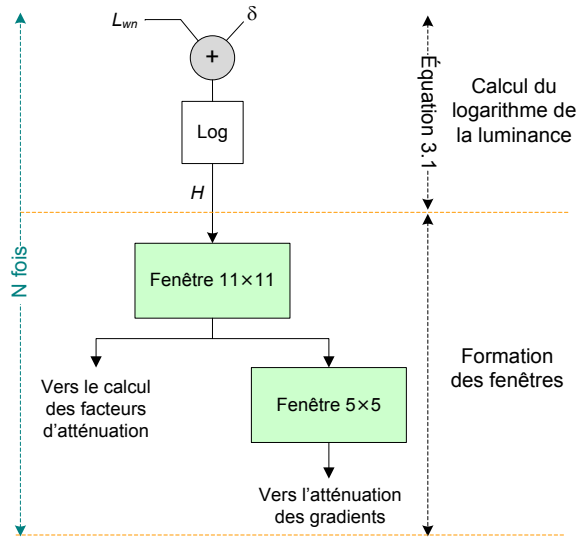


Figure 3.5: Flot du calcul du logarithme et de la formation des fenêtres

Le calcul de la matrice d'atténuation est le bloc le plus important et complexe de cet algorithme. La Figure 3.6 et la Figure 3.7 représentent les opérations requises pour obtenir les facteurs d'atténuation. Dans ces figures, la variable *I* correspond aux images de chaque niveau de la pyramide gaussienne : *H* pour le niveau 0, et *gBlur* (le résultat du filtre gaussien) pour les niveaux postérieurs.

Nous allons maintenant décrire les opérations requises pour les trois premières étapes de la Figure 3.6 : le calcul des gradients centraux, de leur norme et de la moyenne. Deux sommes et deux soustractions sont nécessaires. Le même calcul est répété 16 fois, puisque cette fenêtre possède 4×4 pixels. De plus, tel qu'illustré à la Figure 3.4, les gradients doivent s'exécuter 5 fois (*d* vaut 4 dans ce cas) pour chacun des pixels de l'image. Ainsi, cette étape nécessite 320×*N* sommes/soustractions (4×16×5).

Tel que décrit à la section précédente, une banque de filtres gaussiens est requise pour le calcul de la fonction d'atténuation. La Figure 3.7 montre le flot de calcul du filtre basé sur une convolution avec le noyau choisi (3.2). Dans ce cas, le calcul de la convolution a besoin de 4 multiplications par 2, une multiplication par 4, 8 sommes et une division par 16. Ce même

processus est répété pour chaque niveau de la technique multipasse (voir la Figure 3.4), à savoir : 9×9 pixels, 7×7, 5×5 et 3×3, ce qui donne comme résultat 164 fois pour chaque pixel de l'image.

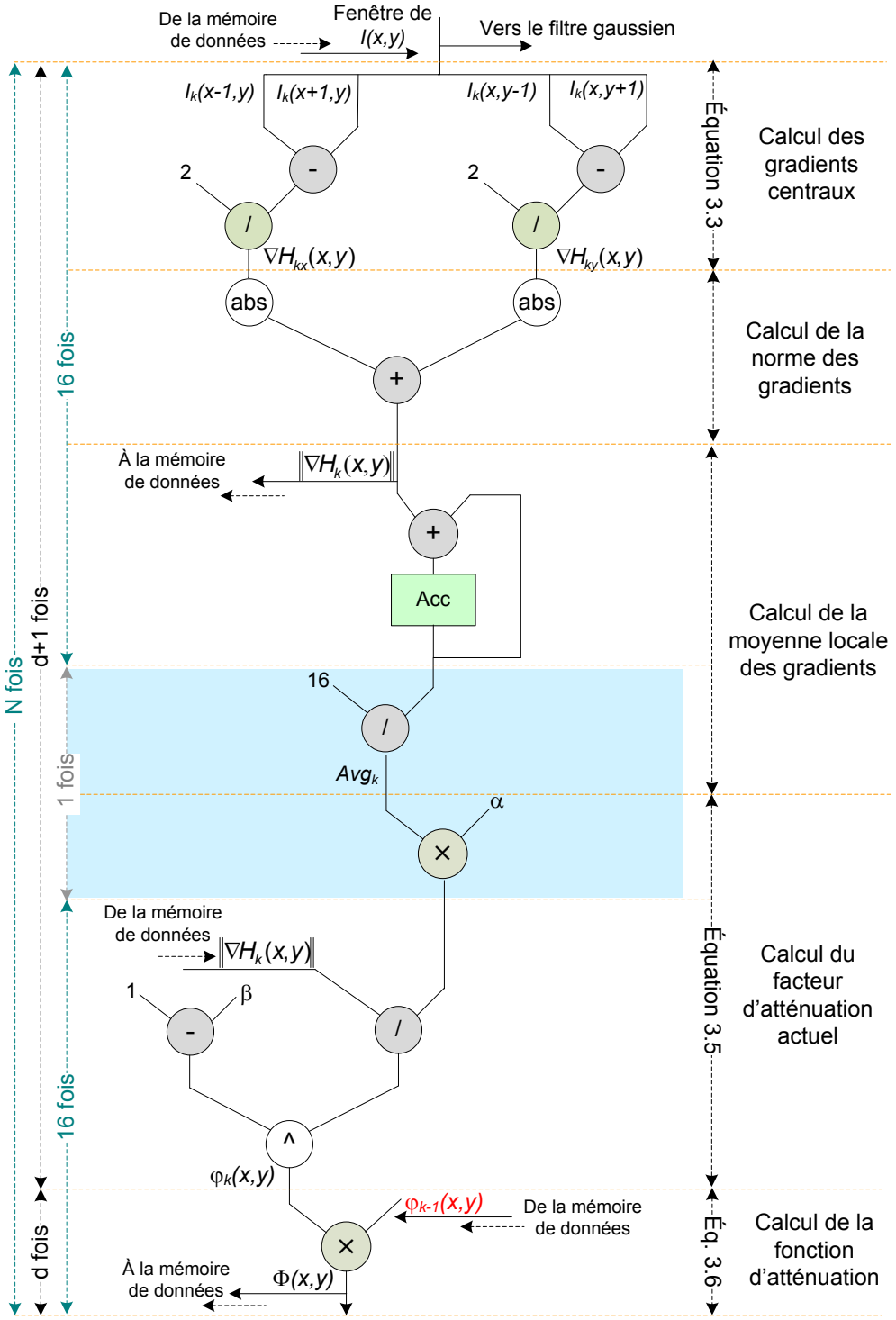


Figure 3.6: Flot de calcul des facteurs d'atténuation

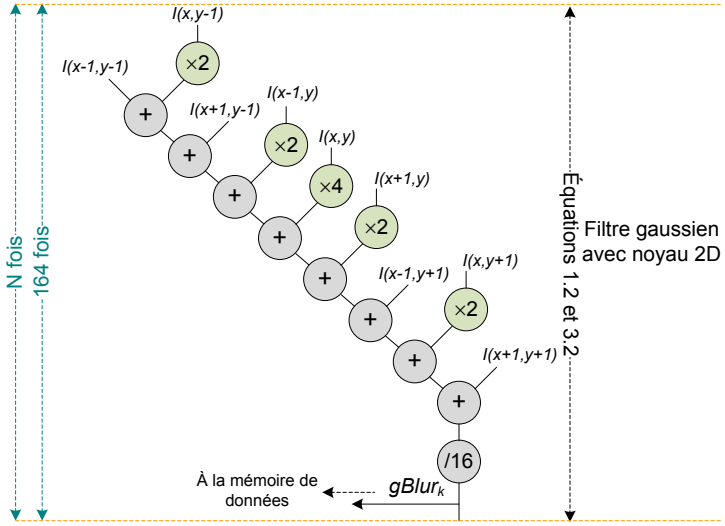


Figure 3.7: Flot de calcul du filtre gaussien

L'analyse précédente se fait pour le reste de l'algorithme. Le Tableau 3.1 résume l'estimation des besoins pour la matrice d'atténuation locale. Il faut noter que la norme des gradients est approximée avec la somme des valeurs absolues (norme 1) afin de simplifier l'implémentation. Le nombre d'opérations pour le calcul des facteurs d'atténuation courants correspond au pire des cas, lorsque l'équation (3.5) est toujours effectuée. De plus, les instructions de chargement, de sauvegarde et de contrôle sont exclues dans cette estimation. À partir de ce tableau, nous remarquons que le filtre gaussien inclut 1,312 sommes, 820 multiplications et 164 divisions. Donc, c'est une fonction candidate pour atteindre une accélération significative.

La Figure 3.8 montre le flot de calcul pour l'atténuation des gradients. Le calcul des gradients, qui sont approximés par des différences-avant négatives, requiert deux soustractions pour prendre en compte les deux composantes des gradients (horizontale et verticale). L'atténuation des gradients nécessite deux multiplications pour chaque gradient. Les mêmes calculs sont répétés pour chaque valeur de la matrice 4×4 et pour les N pixels de l'image.

La Figure 3.9 et la Figure 3.10 indiquent les opérations nécessaires pour reconstruire la luminance. Le calcul de la divergence requiert trois sommes/soustractions pour chaque valeur de la matrice 3×3. Puis, la solution de l'équation de Poisson nécessite 8 sommes/soustractions par pixel. Cependant, la multiplication par 3 de la valeur de la divergence $divG_{2,2}$ équivaut à multiplier $divG_{2,2}$ par 2, et ensuite sommer ce résultat avec la valeur de $divG_{2,2}$. Donc, neuf

sommes/soustractions et une multiplication par 2 sont nécessaires. De plus, deux divisions par 8 et une division par 16 sont requises. Finalement, la luminance est obtenue avec une exponentiation de base 2 et une soustraction pour chaque pixel de l'image.

Tableau 3.1: Estimation des opérations requises pour la matrice d'atténuation

Étape	**Opération**	**Nombre de fois (× Taille de l'image)**
Gradients centraux et moyenne	Somme/soustraction	320
	Valeur absolue	160
	Division/2	160
	Division/16	5
Facteurs d'atténuation actuels φ_k (si l'équation (3.5) est toujours effectuée)	Multiplication	5
	Division	80
	Puissance	80
Matrice d'atténuation finale Φ	Multiplication	64
Filtre gaussien	Somme/soustraction	1312
	Multiplication×2	656
	Multiplication×4	164
	Division/16	164

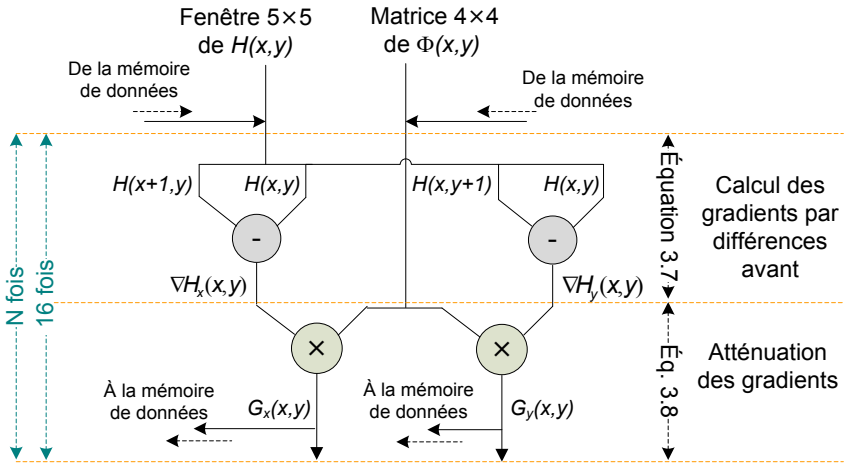


Figure 3.8: Flot de calcul de l'atténuation des gradients

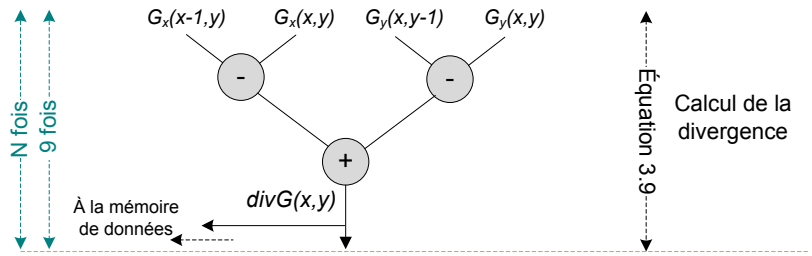


Figure 3.9: Flot de calcul de la divergence des gradients atténués

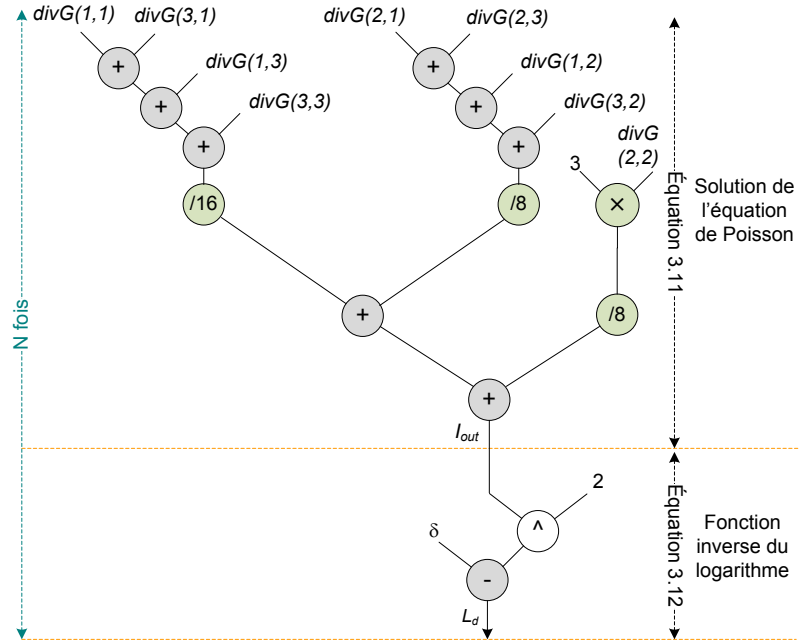


Figure 3.10: Flot de calcul de la reconstruction finale de l'image

Le Tableau 3.2 résume l'estimation des opérations requises pour chaque module principal de l'algorithme de Vytla. Comme on l'a mentionné précédemment, l'étape de la matrice d'atténuation est la plus exigeante en termes du nombre d'opérations. Par ailleurs, le Tableau 3.3 présente les besoins totaux en calculs, toutes étapes confondues. Les valeurs y sont organisées par nombre d'exécutions en ordre décroissant. Même si le logarithme et sa fonction inverse doivent s'exécuter une seule fois par pixel, il faut remarquer que leur calcul comprend une série d'opérations qui dépendent de la méthode utilisée pour leur approximation.

Tableau 3.2: Estimation des opérations requises par étape pour la solution de Vytla

Étape	**Opération**	**Nombre de fois (× Taille de l'image)**
Logarithme	Somme/soustraction	1
	Logarithme	1
Matrice d'atténuation	Somme/soustraction	1632
	Multiplication	69
	Multiplication×2	656
	Multiplication×4	164
	Division	80
	Division/2	160
	Division/16	169
	Puissance	80
	Valeur absolue	160
Atténuation	Somme/soustraction	32
	Multiplication	32
Reconstruction de l'image	Somme/soustraction	37
	Multiplication×2	1
	Division/8	2
	Division/16	1
	Fonction exponentielle	1

Tableau 3.3: Estimation des opérations totales requises pour la solution de Vytla

Opération	**Nombre de fois (× Taille de l'image)**
Somme/soustraction	1702
Multiplication×2	657
Multiplication×4	164
Division/2	160
Valeur absolue	160
Multiplication	101
Division	80
Puissance	80
Division/8	2
Division/16	1
Logarithme	1
Fonction exponentielle	1

En ce qui concerne les besoins en mémoire, l'analyse suivante correspond aux modules principaux de la solution de Vytla, montrés à la Figure 3.1, et à notre calcul de la pyramide gaussienne modifiée. D'abord, il faut une fenêtre 11×11 des valeurs du logarithme, tel qu'illustré à la Figure 3.4. À chaque fois que les filtres gaussiens sont calculés, les valeurs de la fenêtre 11×11 sont remplacées par les valeurs du niveau courant. Donc, il ne faut qu'une matrice 11×11. Après, une fenêtre 6×6 est requise pour le calcul des gradients centraux qui forment une fenêtre 4×4. Par la suite, le calcul de la fonction d'atténuation locale doit enregistrer 4×4 valeurs. En outre, deux matrices 4×4 sont nécessaires pour les gradients par différences avant (voir la Figure 3.2). Ces deux dernières matrices peuvent aussi sauvegarder les résultats de l'atténuation des gradients. La fenêtre de la divergence contient 3×3 valeurs. Finalement, une variable scalaire enregistre le résultat de la solution de l'équation de Poisson. En somme, les modules principaux requièrent de sauvegarder seulement 231 données. Comme chaque donnée est de 32 bits, cela implique 924 octets pour n'importe quelle résolution d'image.

3.3.2 Profilages avec le processeur de base

Après l'estimation manuelle précédente et une fois que l'algorithme de Vytla a été codé en langage C, nous avons analysé les modules principaux de l'algorithme à l'aide des profilages sur le processeur Xtensa de base. Un premier code a été développé en virgule flottante, car le compilateur de Xtensa est capable d'émuler les opérations en virgule flottante (format IEEE754) en logiciel [50]. Par la suite, un deuxième code a été développé, produit de la conversion en virgule fixe.

Les résultats du profilage de l'algorithme de Vytla en virgule flottante sont montrés à la Figure 3.11 et à la Figure 3.12. La Figure 3.11 indique le pourcentage de cycles de chaque fonction des modules principaux. Les opérations de base du processeur y sont comprises, comme la somme, la division, etc. Nous observons que la multiplication occupe 47% de l'effort de calcul, ce qui correspond à la somme des pourcentages de la multiplication en double précision (_muldf3) et de celle en simple précision (_mulsf3). Ces deux types de multiplication appartiennent à l'ensemble de fonctions en virgule flottante émulées par le processeur Xtensa. La multiplication en double précision consomme la plupart des cycles, environ 34%. Cette fonction est appelée par la puissance et le logarithme (voir la Figure 3.12, à droite). De plus, la multiplication en simple précision est la deuxième fonction la plus complexe et prend 13% du total des cycles. Celle-ci se

trouve en plusieurs endroits dans le code : pour le calcul du filtre gaussien, de la matrice d'atténuation, de l'atténuation des gradients, des gradients centraux et de la solution de l'équation de Poisson. Par ailleurs, nous remarquons qu'effectivement la solution de l'équation de Poisson n'est plus une fonction critique en temps.

Function Name	Total cycles (%)	Function cycles	Children cycles	Times called (invocations)
__divdf3	7.57	3975306049	0	7546754
__ieee754_pow	6.52	3425634360	789012121	3724225
__adddf3	6.27	3292395159	41258851	112176682
__addsf3	6.02	3161865493	0	100319233
gaussianBlur(float (*)[11], int, float...	5.19	2727051767	2333323121	196608
__mulsf3	12.73	2389256259	0	120176640
__floatsisf	3.82	2006581261	0	95551488
__subdf3	11.23	1604405536	544530	85593121
__muldf3	33.94	637146938	0	130935091
__subsf3	1.11	584594603	544530	10321920
__divsf3	1.07	562286169	0	3675073
pow	0.51	271868533	179836661	3724225
centralGradients(float (*)[5], float (...	0.48	252887312	1353661911	245760
calcPhiMatrix(float (*)[11], int, floa...	0.42	222368595	3420665550	49152
__adddf3_aux	0.41	216731630	0	<spontaneous>
__extendsfdf2	0.35	188960802	0	11686849
__truncdfsf2	0.33	174041626	0	7705537
getWindow(float **, int, int, unsig...	0.26	138252506	0	49152
isnan	0.21	112464057	0	7497602
__floatsidf	0.12	67743340	0	3770302
__subdf3_aux	0.12	66038525	0	<spontaneous>
__nedf2	0.11	62629720	0	7448450
__addsf3_aux	0.11	61029210	0	<spontaneous>
__muldf3_aux	0.11	59449372	0	<spontaneous>
__nesf2	0.08	47185920	0	3932160
finite	0.08	44690770	0	3724225
_WindowOverflow8	0.08	42347924	0	3822530
_WindowUnderflow8	0.08	42195358	0	3822530
__lesf2	0.07	38807426	0	<spontaneous>
__mulsf3_aux	0.07	38366883	0	<spontaneous>
fabs	0.07	37242250	0	3724225
getSmallWindow(float (*)[11], floa...	0.06	35684756	0	294912
attenuateGrad(float (*)[4], float (*)...	0.05	28311657	176646738	49152
__gtsf2	0.05	27525120	0	3932160
__subsf3_aux	0.04	22987530	0	<spontaneous>
calcDivergence(float (*)[3], float (*...	0.02	15679655	64097354	49152
__ieee754_log	0.02	14083008	131427912	49152
setValue(float *, int, float)	0.02	13566114	0	98305
tmo_fattal02(unsigned int, unsign...	0.0	4032388	443458727	<spontaneous>
localPoisson(float (*)[2])	0.0	3047505	20594898	49152
log	0.0	2064528	147134846	49152
logLumi(float *&, float **, int, int)	0.0	1553773	178456140	1
__ledf2	0.0	540685	0	<spontaneous>
__gtdf2	0.0	344077	0	49152
__divdf3_aux	0.0	196634	0	<spontaneous>
xt_iss_profile_enable	0.0	4	0	<spontaneous>
xt_iss_profile_disable	0.0	4	0	1

Figure 3.11: Résultats du profilage du code de base en virgule flottante

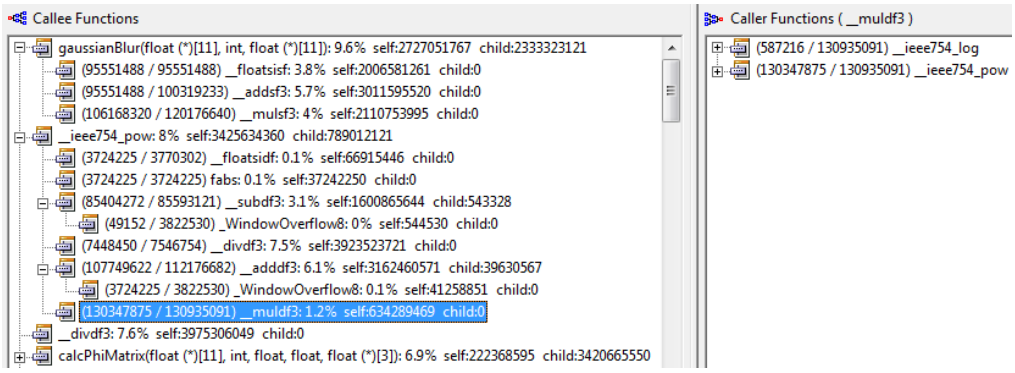


Figure 3.12: Graphe d'appel du code de base en virgule flottante

Le deuxième profilage avec le processeur de base concerne le code en virgule fixe. La Figure 3.13 montre les résultats en termes du nombre de cycles pour les modules principaux de la solution de Vytla. Grâce à la conversion en virgule fixe, la multiplication n'est plus la fonction la plus importante en termes du nombre de cycles exécutés. Cela vient du fait que, dans la mesure du possible, les multiplications ont été remplacées par des décalages. Nous observons que la fonction critique est maintenant le calcul des filtres gaussiens prenant 36% des cycles totaux. Ceci est cohérent avec l'analyse manuelle expliquée précédemment. Les autres deux fonctions les plus complexes, le logarithme et la division, représentent environ 17% et 15% du nombre total de cycles, respectivement.

Function Name	Total cycles (%)	Function cycles	Children cycles	Times called (invocations)
gaussianBlur(int (*)[10], int, int (*)[10])	35.69	651167198	0	196608
log2Fix(unsigned int, int)	17.23	314334799	540742	3723369
__udivsi3	14.55	265439004	0	3674217
calcPhiLocal(unsigned int (*)[3], unsigned in...	7.76	141547609	701034588	245760
exp2Fix(int, int)	6.93	126547243	0	3723369
getWindow(int **, int, int, unsigned int, unsi...	6.13	111900544	0	49152
centralGradients(int (*)[5], unsigned int (*)[3...	4.54	82821336	0	245760
calcPhiMatrix(int (*)[10], int, unsigned int, u...	2.32	42468125	1617317962	49152
getSmallWindow(int (*)[10], int *, int, int *)	2.08	38043927	0	294912
attenuateGrad(int (*)[4], unsigned int (*)[3], i...	1.37	25018601	0	49152
calcDivergence(int (*)[3], int (*)[3], int (*)[2])	0.52	9486362	0	49152
setValue(int *, int, int)	0.46	8503459	0	49153
tmo_fattal02Fix(unsigned int, unsigned int, ...	0.18	3442346	1820601119	<spontaneous>
localPoisson(int (*)[2])	0.07	1376269	0	49152
logLumi(unsigned int *&, int **, int, int)	0.04	865226	4156655	1
_WindowOverflow8	0.02	540742	0	49152
_WindowUnderflow8	0.02	540672	0	49152
xt_iss_profile_enable	0.0	4	0	<spontaneous>
xt_iss_profile_disable	0.0	4	0	1

Figure 3.13: Résultats du profilage du code de base en virgule fixe

3.4 Instructions spécialisées proposées

Grâce à l'analyse manuelle précédente, nous avons une vision générale de la complexité de l'algorithme et des sections méritant potentiellement d'une accélération. Cependant, il faut prendre en compte que l'implémentation du logarithme, de la fonction exponentielle et de la puissance peuvent générer une influence remarquable dans le nombre de cycles. En vue de bien cibler les goulots d'étranglement de l'algorithme de Vytla, nous avons profilé le code non accéléré en virgule fixe sur le processeur Xtensa de base (voir la Figure 3.13). Étant donné que le filtre gaussien représente environ 36% du nombre total de cycles, nous avons concentré nos efforts sur ce module.

Les instructions spécialisées ajoutées utilisent essentiellement trois techniques pour l'amélioration de la performance du calcul du filtre gaussien : fusion d'opérateurs, SIMD (*Single-Instruction Multiple-Data*) et réutilisation de données. De plus, le choix des coefficients du noyau est judicieux, car ils rendent possible le remplacement des multiplications par des décalages. Le principe est d'effectuer des accumulations incluant les décalages respectifs en fonction du noyau, d'accumuler à nouveau en réutilisant quelques résultats précédents et de décaler cette deuxième accumulation pour approximer la division par 16. Ainsi, nous obtenons quatre résultats en parallèle. Ce processus est répété en déplaçant la fenêtre de manière verticale et en recommençant avec les données des colonnes suivantes jusqu'à la fin de la partie ciblée de l'image. D'ailleurs, nous profitons des résultats de quelques accumulateurs pour ne pas les recalculer lors du déplacement vertical.

Nous avons créé deux instructions spécialisées pour accélérer le calcul du filtre gaussien : *filter* et *addShift.* Prenons par exemple une partie *P* de l'image à filtrer :

$$P = \begin{bmatrix} P_{0,0} & P_{0,1} & P_{0,2} & P_{0,3} \\ P_{1,0} & P_{1,1} & P_{1,2} & P_{1,3} \\ P_{2,0} & P_{2,1} & P_{2,2} & P_{2,3} \\ P_{3,0} & P_{3,1} & P_{3,2} & P_{3,3} \end{bmatrix}. \tag{3.13}$$

Le but est de calculer la convolution entre les pixels $P_{1,1}$, $P_{1,2}$, $P_{2,1}$ et $P_{2,2}$ et le noyau gaussien (3.2). Définissons la convolution partielle de la première ligne de $P_{1,1}$ à partir de (3.14) :

$$C0 = P_{0,0} + \left(P_{0,1} \ll 1\right) + P_{0,2}, \tag{3.14}$$

où le symbole ≪ indique un décalage d'un bit vers la gauche. En vue d'obtenir le résultat de $P_{1,1}$, il est possible de calculer la convolution partielle de la troisième ligne avec les mêmes opérations de la première ligne grâce à la symétrie du noyau. De plus, ce noyau permet aussi d'effectuer la convolution de la deuxième ligne en appliquant (3.14) et en doublant ce résultat. La Figure 3.14 montre le principe expliqué précédemment.

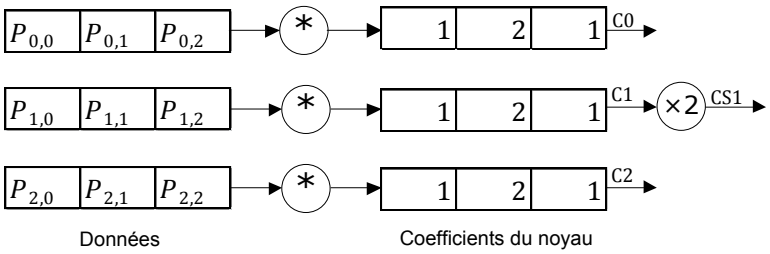


Figure 3.14: Calcul initial de la convolution de $P_{1,1}$ avec le noyau 2D

Pour obtenir la convolution du pixel $P_{1,2}$, les opérations montrées à la Figure 3.14 sont aussi requises, mais avec les données décalées d'une colonne vers la droite. Ainsi, la convolution partielle de la première ligne (3.14) est calculée avec les valeurs $P_{0,1}$, $P_{0,2}$ et $P_{0,3}$. Un processus similaire est aussi valable pour les résultats de $P_{2,1}$ et de $P_{2,2}$. L'instruction spécialisée *filter* est donc définie comme suit :

$$\begin{bmatrix} A0 \\ A1 \\ A2 \\ A3 \end{bmatrix} = \begin{bmatrix} vec1(0) + (vec1(1) \ll 1) + vec1(2) \\ vec1(1) + (vec1(2) \ll 1) + vec1(3) \\ vec2(0) + (vec2(1) \ll 1) + vec2(2) \\ vec2(1) + (vec2(2) \ll 1) + vec2(3) \end{bmatrix}, \tag{3.15}$$

où *vec1* et *vec2* correspondent aux deux lignes d'entrée. Cette instruction doit s'exécuter une deuxième fois pour obtenir la convolution partielle de la troisième ligne.

Pour finir le calcul de la convolution, il reste à multiplier par 2 le résultat de *filter* pour la deuxième ligne (par exemple, CS1 de la Figure 3.14), et de diviser par 16 la somme de toutes les convolutions partielles. C'est pourquoi l'instruction *addShift* contient les opérations définies en (3.16) et en (3.17). L'équation (3.16) effectue l'approximation de la multiplication par 2 pour obtenir la convolution partielle de la deuxième ligne. Après, l'équation (3.17) consiste à accumuler les résultats précédents (par exemple, C0, CS1 et C2 de la Figure 3.14) et à effectuer l'approximation de la division par 16 de cette somme. De cette façon, les valeurs de *R0*, *R1*, *R2* et

R3, qui correspondent aux résultats de $P_{1,1}$, $P_{1,2}$, $P_{2,1}$ et $P_{2,2}$ respectivement, sont obtenues simultanément.

$$\begin{bmatrix} S0 \\ S1 \\ S2 \\ S3 \end{bmatrix} = \begin{bmatrix} vec1(2) \ll 1 \\ vec1(3) \ll 1 \\ vec2(0) \ll 1 \\ vec2(1) \ll 1 \end{bmatrix} \tag{3.16}$$

$$\begin{bmatrix} R0 \\ R1 \\ R2 \\ R3 \end{bmatrix} = \begin{bmatrix} (vec1(0) + S0 + vec2(0)) \gg 4 \\ (vec1(1) + S1 + vec2(1)) \gg 4 \\ (vec1(2) + S2 + vec2(2)) \gg 4 \\ (vec1(3) + S3 + vec2(3)) \gg 4 \end{bmatrix} \tag{3.17}$$

D'un autre côté, il a fallu ajouter une autre instruction pour les itérations postérieures. Cette instruction s'occupe de lire les résultats du filtre au niveau actuel et de les organiser de façon à former la fenêtre attendue à l'entrée du niveau postérieur.

CHAPITRE 4 RÉSULTATS

Dans ce chapitre, nous présentons les résultats de nos expériences de deux points de vue complémentaires : la qualité d'image et les coûts en calcul. Ces deux critères sont importants pour valider les implémentations d'algorithmes de traitement d'images. La section 4.1 comprend les résultats des comparaisons de quatre méthodes d'amélioration du contraste, dont deux algorithmes de reproduction de tons, tel que décrit à la section 2.1. Le reste du chapitre contient les résultats de l'accélération de l'algorithme global de Reinhard et de l'algorithme de Fattal modifié par Vytla, aux sections 4.2 et 4.3, respectivement.

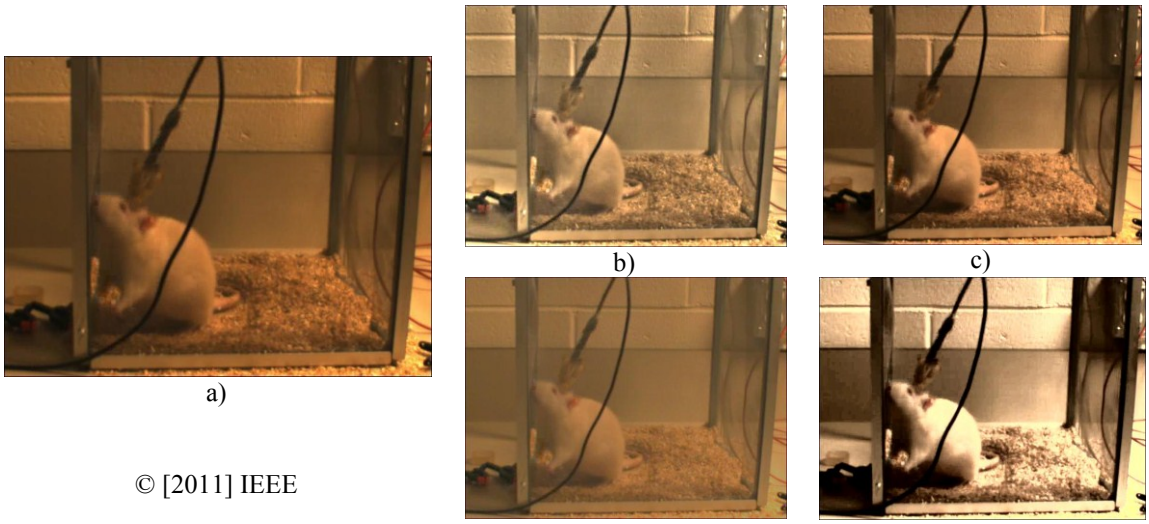
4.1 Résultats expérimentaux de l'amélioration du contraste

Les résultats de cette section correspondent à l'analyse comparative expliquée à la section 2.1. Comme on l'a déjà mentionné, l'amélioration du contraste d'images LDR est une des applications des algorithmes de reproduction de tons. Nous avons comparé deux méthodes de TM : celle de Fattal [2] et la version globale de la méthode de Reinhard [3]; et deux autres algorithmes conçus directement pour ajuster le contraste d'images LDR : celui d'Arici [35], et l'égalisation d'histogramme. Nous les appelons Fattal02, Reinhard02, Arici09 et GHE, respectivement. L'évaluation considérait deux critères : la qualité des images résultantes, tant subjective qu'objective, et le temps d'exécution de chaque algorithme. La qualité est liée à l'amélioration du contraste de toute l'image ainsi que dans quelques régions d'intérêt choisies.

4.1.1 Qualité des images

Nous avons évalué chacun des quatre algorithmes d'amélioration du contraste en termes de la qualité des images. La Figure 4.1 montre une image test originale et la version améliorée résultant de chaque méthode. Nous observons que la méthode de Fattal maintient un bon contraste sur toute l'image (voir la Figure 4.1b), même dans les zones ombragées. Cependant, une vue plus naturelle est obtenue avec la technique d'Arici (voir la Figure 4.1c). Les résultats de l'algorithme global de Reinhard, Figure 4.1d, montrent une image entièrement « brumeuse ». En fait, l'image originale semble avoir un meilleur contraste. Comme on le voit à la Figure 4.1e, le dos du rat est bien éclairé mais ses jambes sont plus sombres que dans l'image originale. Ceci peut indiquer que la méthode d'égalisation d'histogramme (GHE) occasionne que quelques

régions deviennent trop lumineuses et d'autres trop sombres, ce qui a pour résultat des images avec un contraste excessif et un aspect moins naturel.



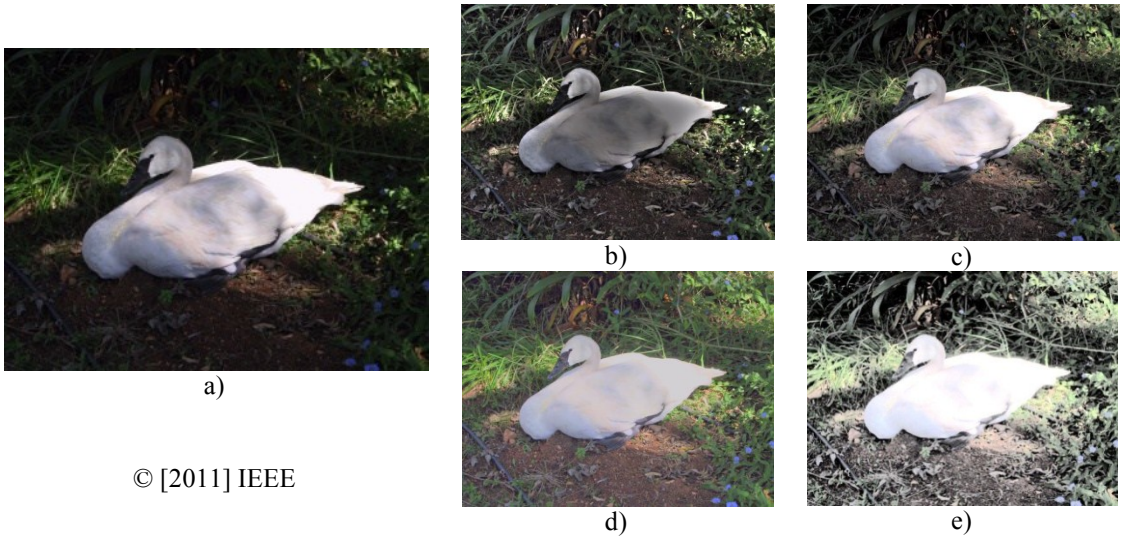
a) b) c) d) e)

Figure 4.1: Rat en position 1, 640×480 pixels. a) Image originale, b) Image améliorée avec Fattal02, c) Arici09, d) Version globale de Reinhard02, e) GHE

La Figure 4.2 montre les résultats pour l'image « Swan ». La méthode de Fattal améliore le contraste dans l'arrière-plan, mais le diminue pour le sujet. Une explication possible de cette réduction est que les paramètres peuvent requérir un ajustement. À la Figure 4.2c, nous pouvons observer qu'Arici09 produit un bon ajustement du contraste, bien que les plantes derrière le cygne soient plus sombres que le résultat de Fattal. Reinhard02 améliore le contraste global et « l'effet de brume » est presque absent (voir la Figure 4.2d). Ces résultats nous amènent à penser que certains algorithmes sont plus sensibles que d'autres aux valeurs de leurs paramètres. GHE donne encore un contraste excessif et un aspect peu naturel de l'image (voir la Figure 4.2e)).

Les résultats des métriques quantitatives pour évaluer la qualité des images sont montrés à la Figure 4.3. Selon les valeurs AMBE et le contraste d'intensités relatif (décrits à la section 1.3.1), le meilleur compromis pour les images des rats est obtenu par Arici09. De plus, bien que Reinhard02 donne parfois des valeurs AMBE plus petites que Fattal02, par exemple dans le cas du Rat2, ceci n'indique pas nécessairement que le contraste de l'image soit amélioré. Cependant, Fattal02 a produit la valeur AMBE la plus petite et un contraste d'intensités élevé pour l'image « Swan ». Cela ne coïncide pas exactement avec l'évaluation subjective présentée précédemment. En outre, GHE obtient la valeur du contraste la plus élevée, ce qui indiquerait que cette méthode

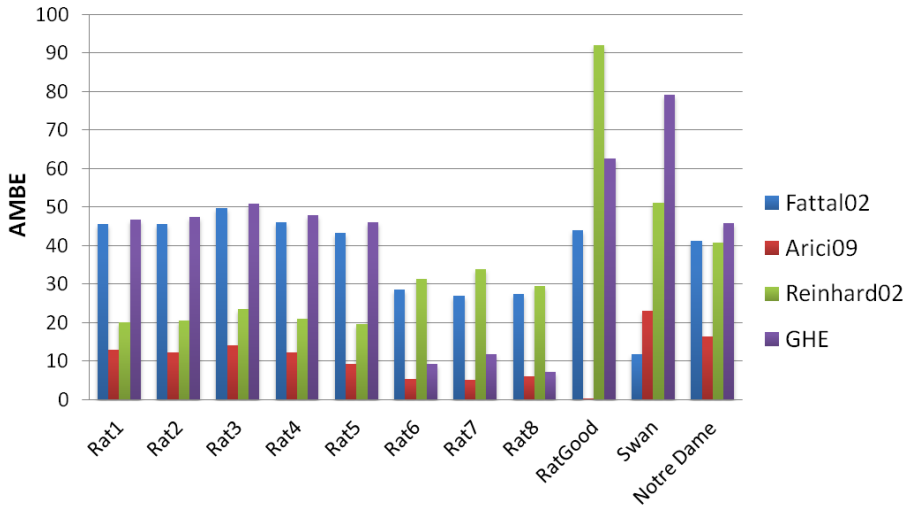
obtient la meilleure qualité. Néanmoins, cette valeur n'est pas en accord avec l'image la plus agréable visuellement. Donc, une valeur AMBE plus petite ou un haut contraste d'intensités ne signifie pas nécessairement que l'image préserve son apparence naturelle.



a)

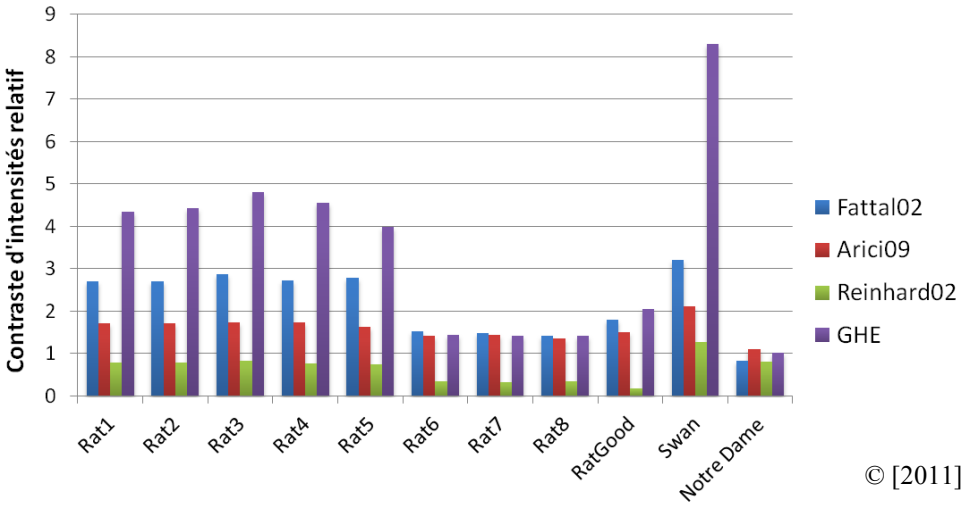
b) c) d) e)

Figure 4.2: Image Swan, 513×385 pixels. a) Image originale tirée de [13], reproduite avec permission, b) Image améliorée avec Fattal02, c) Arici09, d) Version globale de Reinhard02, e) GHE



a)

Figure 4.3: Résultats des métriques de qualité d'images. a) AMBE, b) contraste d'intensités relatif



b)

Figure 4.3 (Continuation) : Résultats des métriques de qualité d'images. a) AMBE, b) contraste d'intensités relatif

Au sujet de l'amélioration du contraste dans des zones d'intérêt ombragées, la Figure 4.4 montre quelques détails d'une image originale et les résultats respectifs de chaque algorithme. Les résultats des métriques objectives sont présentés au Tableau 4.1. De façon subjective, la technique de Fattal fait ressortir plus de détails que les autres méthodes. Cela est cohérent avec le fait que Fattal02 atteint la valeur d'entropie la plus élevée. Cependant, GHE génère un bon niveau de contraste et obtient encore la valeur du contraste la plus grande. Dans ce cas, la technique d'Arici n'améliore pas le contraste de manière significative. Reinhard02 a pour résultat une image floue, ce qui coïncide avec la diminution de la valeur de la métrique du contraste d'intensités.

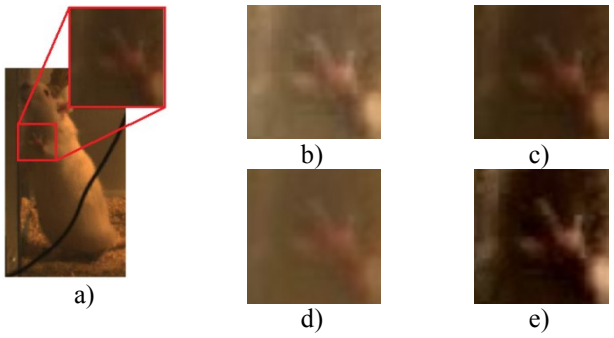


Figure 4.4: Détails de l'image Rat en position 2. a) Image originale, b) Image améliorée avec Fattal02, c) Arici09, d) Version globale de Reinhard02, e) GHE

Tableau 4.1: Résultats de l'entropie et du contraste d'intensités pour les détails de l'image Rat2

Métrique	**Images**				
	Originale	***Fattal02***	***Arici09***	***Reinhard02-Global***	***GHE***
Entropie	6,2	6,8	6,4	6,3	6,5
Contraste d'intensités	5,9	23,3	14,70	4,8	70,7

4.1.2 Complexité des calculs

Les algorithmes de Fattal, d'Arici, la version globale de l'algorithme de Reinhard et l'égalisation d'histogramme (GHE) ont été implémentés en langage C sous l'environnement Cygwin. Toutes les expériences ont été exécutées en utilisant un logiciel sur un fil d'un processeur Intel Pentium à 4 CPUs, 3 GHz et RAM de 2 GB. Les résultats des taux de trame sont présentés à la Figure 4.5. Fattal02 et GHE produisent le débit le plus petit et le plus élevé, respectivement.

S'exécutant au moins 430× plus vite que Fattal02, GHE atteint des taux de trame en temps réel autour de 50 fps pour des images de 640×480 pixels et de 80 fps pour 513×385 pixels. Arici09 s'exécute 340× plus vite que Fattal02, avec des taux de trame d'entre 42 et 64 fps pour ces mêmes résolutions. Par contre, Fattal02 et Reinhard02 ne peuvent pas être utilisés pour le traitement vidéo en temps réel à un taux de trame acceptable avec cette implémentation logicielle. Les taux de trame pour les images Rat6 jusqu'à Rat8 sont inférieurs à cause de leur résolution plus élevée, de 1024×768 pixels.

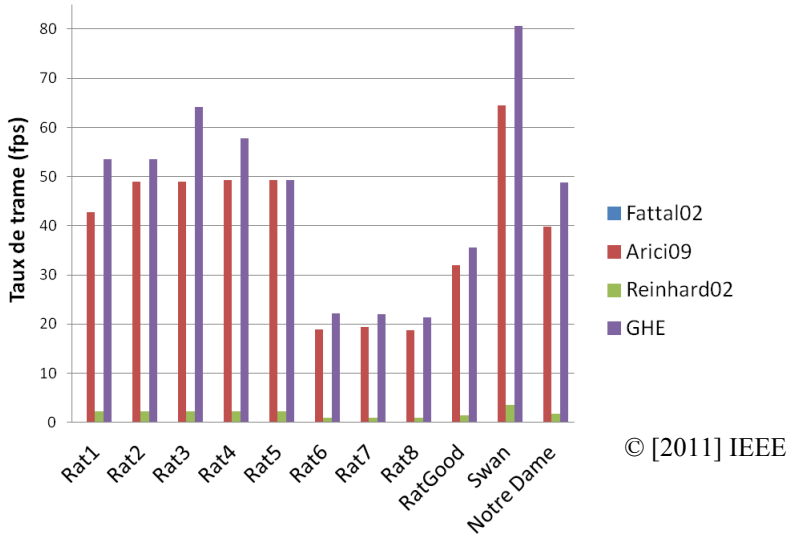


Figure 4.5: Taux de traitement (fps)

4.2 Résultats de l'accélération de l'algorithme global

Les résultats de cette section correspondent à l'accélération de l'algorithme global de reproduction de tons proposé par Reinhard et al. [3], décrit au Chapitre 2. Tel que discuté dans ce dernier, un ASIP a été conçu basé sur le modèle LISA du processeur LTRISC. Nous y avons ajouté trois instructions spécialisées : calculer la luminance, trouver la valeur maximale de la luminance et calculer le logarithme des luminances. L'évaluation des résultats a été effectuée sous différents points de vue : la qualité de l'image, le taux de trame atteint, la surface supplémentaire du processeur et une estimation de l'amélioration de la consommation d'énergie.

Pour l'évaluation du processeur spécialisé proposé pour l'algorithme de Reinhard, les expériences ont été réalisées avec des images test HDR standard, converties à des valeurs RGB en virgule fixe. Nous avons constaté que 10 bits entiers et 22 bits fractionnaires pour les données d'entrée donnent des résultats satisfaisants en termes de la qualité de l'image. Étant donné qu'aucun compilateur C n'était disponible pour notre processeur spécialisé, tous les essais ont été effectués après avoir codé l'application en langage assembleur.

4.2.1 Qualité des images

Nous allons maintenant décrire les résultats obtenus avec l'ASIP conçu pour l'algorithme global de Reinhard et al., en termes de la qualité des images. Tel qu'elle a été expliquée à la section 2.5, une des instructions spécialisées ajoutées porte sur le calcul du logarithme de la luminance. Nous avons comparé deux approximations de la fonction logarithme : la technique de Mitchell et une amélioration de cette technique. Quatre images HDR représentatives ont été utilisées pour mesurer la précision de la conception. Les résultats de l'implémentation matérielle en virgule fixe ont été comparés avec ceux de MATLAB en virgule flottante. Pour ce faire, nous avons mesuré le PSNR entre l'image en virgule fixe et l'image en virgule flottante, tel que défini en (1.8). Comme illustré au Tableau 4.2, lorsque la valeur de l'erreur moyenne du logarithme de Mitchell est ajustée, les résultats du PSNR sont normalement plus grands que 50 dB, ce qui indique un niveau supérieur de qualité [4]. Pourtant, si nous utilisons la version sans correction du logarithme de Mitchell, les images résultantes présentent une qualité moyenne.

Tableau 4.2: Valeurs de PSNR obtenues pour quatre images représentatives

Image	Résolution	Logarithme de Mitchell	Logarithme de Mitchell corrigé
Belgium	256×192	43,7 dB	52,6 dB
Nave	720×480	42,9 dB	50,9 dB
bigFogMap	751×1130	43,9 dB	54,0 dB
Memorial	1024×768	43,9 dB	54,3 dB

Nous avons aussi évalué la qualité des images LDR en prenant comme référence les images HDR originales. Comme décrit à la section 1.3.2, la métrique DRI fournit un plan de distorsions pour détecter des pertes de contraste, des ajouts de contraste qui n'existait pas dans l'image de référence et des renversements du contraste. Le code couleur utilisé est vert, bleu et rouge, pour chaque type de distorsion, respectivement. Le plan montre à chaque pixel la distorsion ayant la probabilité de détection la plus élevée. La Figure 4.6 contient les images résultantes avec l'implémentation en virgule fixe (après une correction gamma de 1,6). La Figure 4.7 montre les plans de distorsions respectifs. L'outil d'évaluation est disponible en ligne [54].

Selon les résultats de la métrique DRI, l'image qui présente le plus grand nombre de distorsions est « Belgium ». La distorsion la plus grave est à cause de la « création » de contraste invisible dans l'image de référence. Ce type de distorsion ne semble pas être en accord avec la nature globale de l'algorithme, car l'amplification du contraste inexistant est associée à l'amélioration de la visibilité de détails propre des algorithmes locaux. L'image « bigFogMap » présente principalement une perte de contraste concentrée dans une région ainsi que distribuée un peu partout dans l'image. Pour les images « Nave » et « Memorial », les résultats sont assez fidèles aux images originales. Toutefois, il y a un manque du contraste dans plusieurs régions et de forts renversements du contraste dans d'autres.

a) Belgium

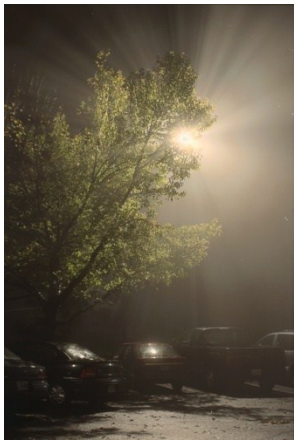
Image HDR originale tirée de [13].

Reproduite avec permission.

b) Nave

Image HDR originale tirée de [55].

Reproduite avec permission.



c) bigFogMap

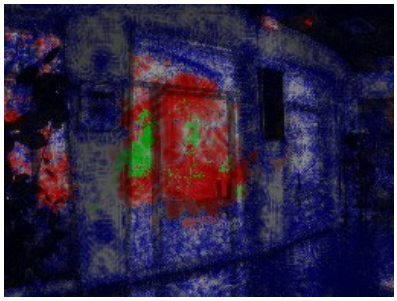
Image HDR originale gracieuseté de Jack Tumblin, Northwestern University [56].

d) Memorial

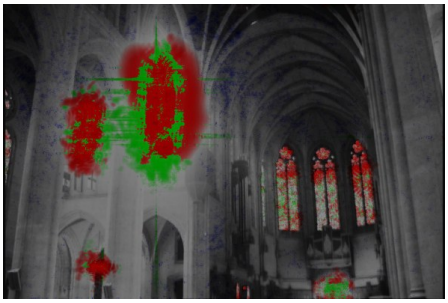
Image HDR originale tirée de [55].

Reproduite avec permission.

Figure 4.6: Images résultantes avec l'algorithme de Reinhard en virgule fixe

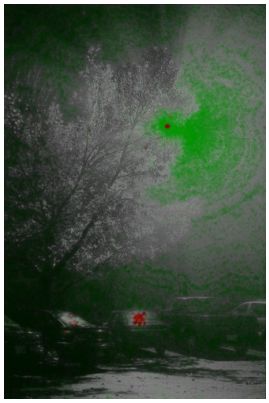


a) Belgium

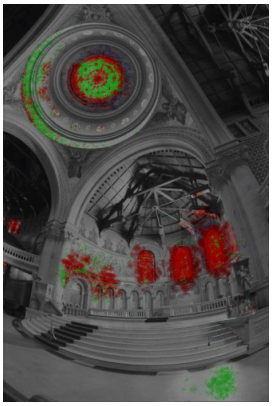


b) Nave

Figure 4.7: Résultats de la métrique DRI pour l'algorithme de Reinhard en virgule fixe



c) BigFogMap



d) Memorial

Figure 4.7 (Continuation) : Résultats de la métrique DRI pour l'algorithme de Reinhard en virgule fixe

4.2.2 Performance, surface supplémentaire et efficacité énergétique

Selon l'étude de performance de [32], le temps d'exécution de la plupart d'algorithmes de reproduction de tons est une fonction de la taille de l'image. À priori, la méthode globale de Reinhard et al. semble suivre un comportement linéaire. Pour le vérifier, nous avons fait une expérience avec l'image « Belgium » à trois résolutions : 256×192, 512×384 et 1024×768 pixels. Ce test a été fait en virgule flottante sur un processeur Intel Core 2 Duo à 1,5 GHz. La Figure 4.8 montre qu'en effet la relation est linéaire. Cela nous facilitera les comparaisons avec d'autres implémentations via le temps normalisé (voir la section 5.2).

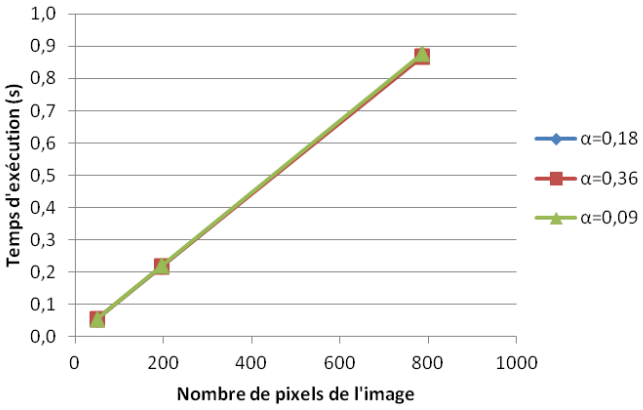


Figure 4.8: Relation entre le temps d'exécution et la résolution de l'image (algorithme global de Reinhard)

Pour le processeur spécialisé LTRISC, les résultats de la performance en termes du nombre de cycles et des taux de trame sont au Tableau 4.3. Sans perte de généralité, ces résultats ont été obtenus avec l'image test « Belgium ». Il faut remarquer que les instructions spécialisés (CI) pour le calcul du logarithme, Log CI, fournit une amélioration de la performance de 139% par rapport

au processeur de base. Par contre, Max CI et Lum CI sont dans le même ordre du nombre de cycles obtenus avec le processeur de base. Lorsque toutes les CI sont ajoutées, le processeur présente une accélération de 169% et atteint 25 fps à une fréquence d'horloge de 85 MHz. Cette amélioration est obtenue principalement par Log CI. Cependant, la valeur atteinte est légèrement plus élevée qu'au cas où les améliorations individuelles sont multipliées. Ces différences peuvent provenir du calcul de la luminance, étant donné que les constantes requises ne sont pas chargées de la mémoire lorsque Lum CI est utilisée.

Tableau 4.3: Résultats de la performance pour un FPGA Virtex-5 XC5VLX30 (Belgium, 256×192 pixels)

Configuration du processeur	**Nombre de cycles totaux**	**Fréquence maximale (MHz)**	**Taux de trame (fps)**	**Amélioration de la performance (fps)**
LTRISC de base	9.121.175	85,51	9,4	0%
Lum CI	8.777.111	85,26	9,7	4%
Log CI	3.784.764	84,71	22,4	139%
Max CI	8.973.719	85,49	9,5	2%
Lum + Log CIs	3.391.614	83,95	24,8	164%
Log + Max CIs	3.637.307	84,68	23,3	148%
Toutes les CIs	3.293.244	83,11	25,2	169%

Le processeur spécialisé proposé pour l'algorithme global de Reinhard a été implémenté sur un FPGA Virtex-5 de Xilinx. Le résumé des résultats se trouve au Tableau 4.4. Nous remarquons que l'instruction spécialisée de la luminance présente une augmentation importante en surface, 17% par rapport à 4% pour Log CI et 2% pour Max CI. Ceci peut s'expliquer par le fait que Lum CI inclut des multiplications, des sommes et des ressources logiques requises pour utiliser le registre destination comme un autre opérande. L'opération du logarithme est plutôt composée d'opérations au niveau du bit. Le rapport de synthèse indique aussi que la fréquence maximale obtenue est autour de 85 MHz dans tous les cas (voir le Tableau 4.3). En conséquence, l'ajout de ces instructions particularisées n'affecte pas la période d'horloge originale.

La Figure 4.9 illustre les résultats de la performance versus la surface additionnelle. La performance la plus élevée est atteinte lorsque toutes les CI sont ajoutées. Dans ce cas, le nombre de cycles a été réduit de 64% aux dépens de 22% en ressources matérielles supplémentaires. Par conséquent, nous avons diminué potentiellement les besoins en consommation de puissance. La plupart des ressources supplémentaires sont ajoutées par Lum CI. Max CI ajoute la moitié de la surface matérielle de Log CI, alors que la performance n'est pas améliorée considérablement.

Comme montré au Tableau 4.4, une estimation de l'efficacité énergétique a été calculée en divisant le facteur d'accélération par le facteur d'augmentation de surface, connu comme le produit temps-surface (AT). Le produit AT relatif donne une estimation du facteur d'amélioration de la consommation d'énergie [46]. La valeur obtenue pour Log CI indique que cette instruction spécialisée fait preuve d'un bon compromis entre l'accélération et le coût matériel. Dans ce cas, le processeur fonctionne 139% plus vite que le processeur original avec une surface supplémentaire négligeable.

Tableau 4.4: Résultats de la synthèse pour un FPGA Virtex-5 XC5VLX30

Configuration du processeur	# LUT	# DSP48E	Augmentation de la surface (# LUT)	Efficacité énergétique
LTRISC de base	4.155	7	0%	1,0
Lum CI	4.851	11	17%	0,9
Log CI	4.305	7	4%	2,3
Max CI	4.250	7	2%	1,0
Lum + Log CI	5.053	11	22%	2,2
Log + Max CI	4.399	7	6%	2,3
Toutes les CI	5.086	11	22%	2,2

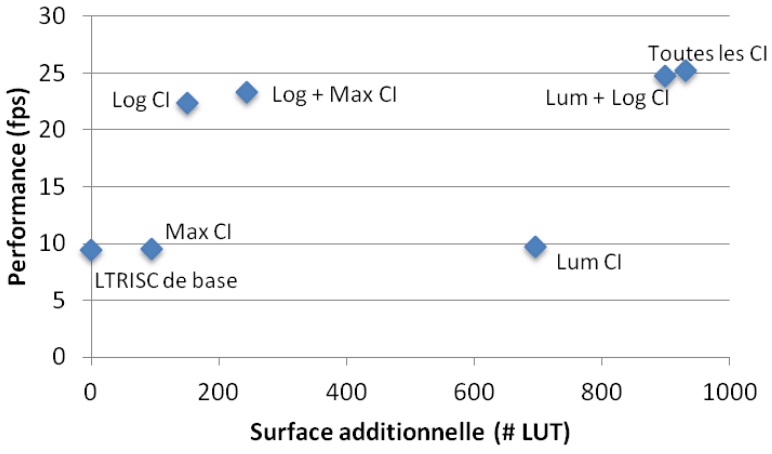


Figure 4.9: Résultats de la performance vs. la surface additionnelle requise

4.3 Résultats de l'accélération de l'algorithme local

Les résultats de cette section correspondent à l'accélération de l'algorithme local de reproduction de tons proposé par Vytla et al. [25], expliqué au Chapitre 3. Un ASIP a été conçu en ajoutant des instructions spécialisées à un processeur Xtensa. Ces instructions accélèrent l'exécution de la banque de filtres gaussiens utilisée pour le calcul de la matrice d'atténuation. Les expériences ont

consisté à évaluer la qualité des images résultantes, la performance, la surface supplémentaire requise et l'efficacité énergétique de l'ASIP proposé. Ces expériences ont été effectuées après avoir codé l'application en langage C.

Un premier code de la solution proposée par Vytla a été développé en virgule flottante, car le compilateur de Xtensa est capable d'émuler les opérations en virgule flottante (format IEEE754) en logiciel [50]. Cependant, cette émulation fait ralentir l'exécution de l'application et peut mener à une consommation élevée de puissance. Par conséquent, nous avons transformé toutes les opérations des modules principaux en virgule fixe. C'est une étape cruciale, étant donné que la qualité de l'image résultante dépend du nombre de bits choisi. C'est aussi une étape complexe, puisqu'il y a de nombreuses variables qui ont besoin de différentes précisions. L'Annexe 2 contient les détails de la méthodologie suivie pour cette conversion.

Afin de faciliter l'implémentation sur le processeur Xtensa, nous avons d'abord simulé l'algorithme sur MATLAB, en virgule flottante et en virgule fixe. Après avoir trouvé le nombre de bits adéquats pour chaque variable, nous avons procédé à implanter le code en virgule fixe sur le processeur Xtensa de base. Finalement, nous avons vérifié les résultats de ce dernier avec les résultats en virgule fixe de MATLAB.

4.3.1 Qualité des images

Dans cette section, nous allons présenter les résultats de l'ASIP proposé pour l'algorithme de Vytla en termes de la qualité des images. Comme mentionné ci-haut, le code de l'application a été d'abord simulé en virgule flottante et, par la suite, en virgule fixe. La Figure 4.10 montre les images résultantes de la simulation sur MATLAB en virgule flottante et en virgule fixe, pour un jeu d'images HDR reconnues. En observant ces images, nous constatons que la reproduction de tons en virgule fixe produit des images de très haute qualité, par rapport aux résultats en virgule flottante.

Afin d'analyser la qualité des images de manière objective, nous avons appliqué plusieurs métriques utilisées dans la littérature. Pour les deux premières métriques, la référence a été l'image obtenue en virgule flottante. La première métrique est la moyenne de l'index de similarité de structure (MSSIM) [39], mentionné précédemment à la section 1.3.1. La deuxième métrique est le PSNR (*Peak Signal to Noise Ratio*), défini à (1.8). Les valeurs du MSSIM et du

PSNR, rassemblées au Tableau 4.5, confirment que les images produites par la simulation en virgule fixe possèdent un niveau haut de qualité.

Belgium House (α = 0,01; β = 0,9). Image HDR originale tirée de [13]. Reproduite avec permission.

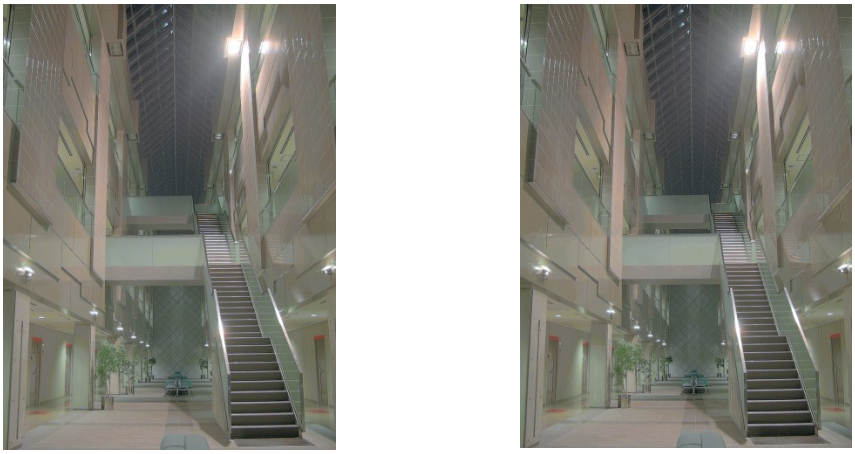
Nave (α = 0,01; β = 0,9). Image HDR originale tirée de [55]. Reproduite avec permission.

Vine Sunset (α = 0,2; β = 0,9). Image HDR originale tirée de [55]. Reproduite avec permission.

a) b)

Figure 4.10: Images résultantes de l'algorithme de Vytla, a) en virgule flottante, b) en virgule fixe

Memorial (α = 0,01; β = 0,9). Image HDR originale tirée de [55]. Reproduite avec permission.



Atrium Night (α = 0,01; β = 0,9)

Image HDR originale tirée de "Image Gallery." [En ligne]. Disponible: http://www.mpi-inf.mpg.de/resources/hdr/gallery.html. [Consulté le 27 janvier 2012]. © Frédéric Drago

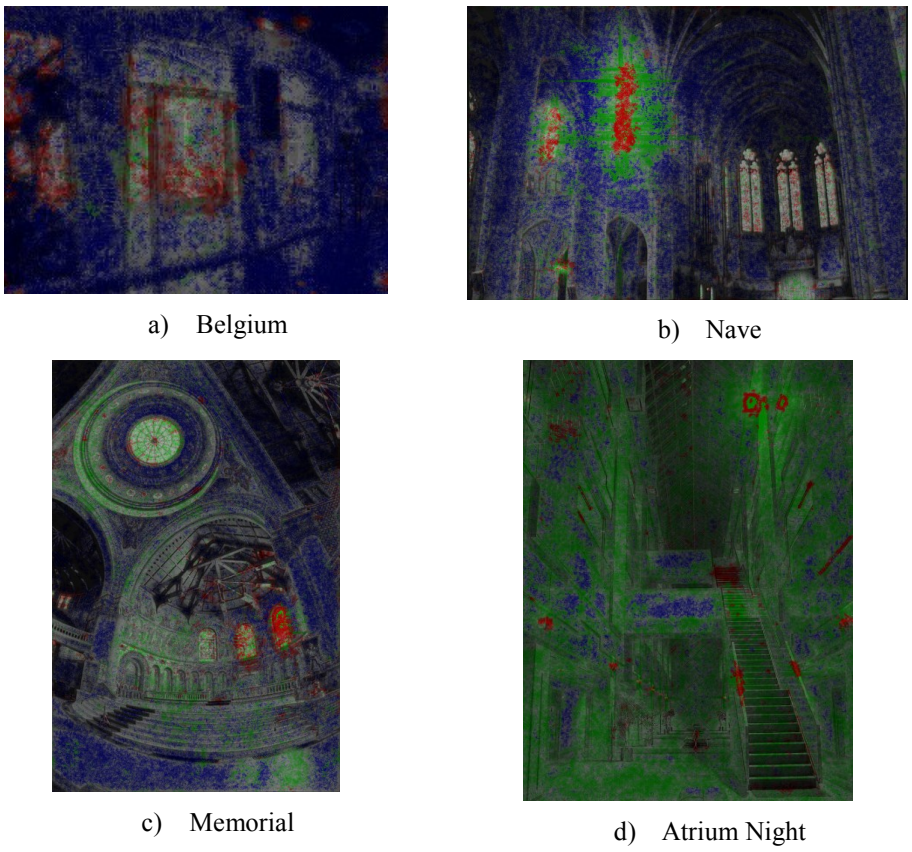
a) b)

Figure 4.10 (Continuation) : Images résultantes de l'algorithme de Vytla, a) en virgule flottante, b) en virgule fixe

Tableau 4.5: Valeurs de MSSIM et de PSNR pour plusieurs images représentatives

Image	**Résolution**	**MSSIM**	**PSNR (dB)**
Belgium House	1024×768	0,99	71
Nave	720×480	1	89
Vine Sunset	720×480	1	84
Atrium Night	760×1016	1	90
Memorial	512×768	1	94

En vue de comparer les images HDR originales et les images LDR résultantes, nous avons utilisé la métrique DRI (voir la section 1.3.2). Comme on l'a dit précédemment à la section 4.2, les pertes du contraste visible sont identifiées en vert, l'amplification du contraste invisible en bleu et le renversement du contraste en rouge. Même si les images obtenues sont considérées de haute qualité de manière subjective, la métrique DRI indique les distorsions une fois qu'elles sont comparées avec les images HDR respectives (voir la Figure 4.11). Cette métrique nous montre que le contraste invisible est beaucoup plus fort dans les images « Belgium » et « Nave ». Par contre, l'image « Atrium Night » présente plus de pertes du contraste visible. Des pertes du contraste sont aussi remarquables dans deux des vitraux de « Nave », ainsi que du renversement du contraste visible. L'image « Memorial » manque aussi de contraste dans beaucoup de régions.



a) Belgium b) Nave

c) Memorial d) Atrium Night

Figure 4.11: Résultats de la métrique DRI pour l'algorithme de Vytla en virgule fixe

Quant à la vérification de la conversion à virgule fixe sur le processeur, nous avons mesuré le nombre de bits les moins significatifs (LSB) différant du modèle en virgule fixe sur MATLAB, pixel à pixel. Nous avons défini une erreur comme un résultat avec une différence supérieure à 2

LSB. Conformément à ce requis, nous avons effectué tous les ajustements au code jusqu'au moment où les erreurs de 2 LSB ont disparu.

4.3.2 Performance, surface supplémentaire et efficacité énergétique

Nous avons évalué la performance, le coût matériel additionnel et l'efficacité énergétique estimée de l'ASIP proposé pour l'algorithme de Vytla. En ce qui concerne la performance, considérée comme l'effort de calcul, nous nous sommes concentrés sur le nombre total de cycles. Toutes les simulations ont été faites précises au cycle d'horloge avec différentes configurations du processeur Xtensa. Une mémoire cache de données de 32 KB, 64 octets par ligne et associativité à 4 voies a été utilisée. Une mémoire cache d'instructions avec les mêmes caractéristiques a aussi été incluse.

Tel que décrit à la section 3.3.2, l'application a été profilée en virgule flottante et en virgule fixe avec le processeur de base. Grâce à la conversion en virgule fixe, la multiplication n'est plus la fonction la plus importante en termes du nombre de cycles exécutés. Cela vient du fait que, dans la mesure du possible, les multiplications ont été remplacées par des décalages. Le goulot d'étranglement est maintenant le calcul des filtres gaussiens prenant à peu près 40% des cycles totaux. Une fois que les instructions spécialisées sont ajoutées, les filtres gaussiens ne représentent qu'environ 4%. La Figure 4.12 le démontre avec la somme des pourcentages des fonctions *gaussian12*, *gaussian10*, *gaussian8* et *gaussian6*. Plus exactement, le calcul des filtres gaussiens passe de 651.167.198 à 53.134.570 cycles, ce qui équivaut à une accélération de 12,3×.

En comparant le nombre de cycles obtenu avec les configurations en virgule fixe incluant les TIE et en virgule flottante, l'accélération atteinte est de 39,2× par rapport au code en virgule flottante et de 1,5× par rapport au code de base en virgule fixe. Le Tableau 4.6 contient les résultats de la performance obtenue pour chaque configuration du processeur Xtensa. Le premier facteur d'accélération est mesuré lorsque le processeur de référence émule les opérations en virgule flottante en logiciel, tandis que pour le deuxième facteur d'accélération le processeur de base réalise toutes les opérations en virgule fixe.

Function Name	Total cycles (%)	Function cycles	Children cycles	Times called (invocations)
log2Fix(unsigned int, int)	25.09	314334831	540742	3723369
__udivsi3	21.18	265439004	0	3674217
calcPhiLocal(unsigned int (*)[3], unsigned in...	11.59	145221835	701034641	245760
getWindow(int **, int, int, unsigned int, unsi...	10.47	131221035	0	49152
exp2Fix(int, int)	10.1	126547264	0	3723369
centralGradients(int (*)[5], unsigned int (*)[3]...	6.61	82821351	0	245760
calcPhiMatrixTIE(int (*)[11], int, unsigned int,...	3.68	46203804	1022959720	49152
getSmallWindow(int (*)[11], int *, int, int *)	3.03	38044039	0	294912
attenuateGrad(int (*)[4], unsigned int (*)[3], i...	1.99	25018666	0	49152
gaussian12(int (*)[11], int, int (*)[11])	1.8	22659726	0	49152
gaussian10(int (*)[11], int, int (*)[11])	1.31	16417265	0	49152
calcDivergence(int (*)[3], int (*)[3], int (*)[2])	0.75	9486362	0	49152
gaussian8(int (*)[11], int, int (*)[11])	0.72	9093204	0	49152
setValue(int *, int, int)	0.67	8503459	0	49153
gaussian6(int (*)[11], int, int (*)[11])	0.39	4964375	0	49152
tmo_fattal02Fix(unsigned int, unsigned int, u...	0.27	3442550	1249299142	<spontaneous>
localPoisson(int (*)[2])	0.1	1376269	0	49152
logLumi(unsigned int *&, int **, int, int)	0.06	865235	4156656	1
_WindowOverflow8	0.04	540742	0	49152
_WindowUnderflow8	0.04	540672	0	49152
xt_iss_profile_enable	0.0	4	0	<spontaneous>
xt_iss_profile_disable	0.0	4	0	1

Figure 4.12: Résultats du profilage du code accéléré sur Xtensa

Tableau 4.6: Résultats de la performance en termes du nombre de cycles

Configuration du processeur	**Nombre de cycles**	**Facteur d'accélération 1 (réf. : virgule flottante)**	**Facteur d'accélération 2 (réf. : virgule fixe)**
De base, virgule flottante	49.110.648.315	1,0×	--
De base, virgule fixe	1.824.043.470	26,9×	1,0×
Virgule fixe avec TIE	1.252.741.696	39,2×	1,5×

Nous allons maintenant présenter une estimation des coûts matériels supplémentaires et de l'amélioration en consommation énergétique pour le processeur spécialisé de l'algorithme de Vytla. Les outils fournis pour le processeur Xtensa incluent un estimateur de surface en termes du nombre de portes logiques. Dans ce cas, le processeur proposé nécessite environ 19% plus de portes logiques (voir le Tableau 4.7). Des 14.932 portes additionnelles, 9.764 (65%) sont utilisées pour les fichiers de registres, 3.394 (22%) pour les opérations et les portes restantes sont principalement pour le décodeur.

Un estimé de l'efficacité énergétique, connu comme le produit temps-surface (AT, *Area- Time*), a aussi été calculé et montré au Tableau 4.7. Le produit AT relatif donne une estimation du facteur d'amélioration de la consommation d'énergie [46]. Son calcul consiste à diviser le facteur d'accélération par le facteur d'augmentation de la surface du processeur. L'Xtensa Xplorer estime que la description TIE ajoutée consomme environ 93.798 pJ/cycle (μW/MHz), pour une technologie de 0,13 micromètres à basse tension.

Tableau 4.7: Résultats estimés de l'utilisation de la surface et de l'efficacité énergétique

Configuration du processeur	**Surface requise (# portes logiques)**	**Augmentation de la surface**	**Efficacité énergétique**
De base	79.000	0%	1,0
Virgule fixe avec TIE	93.932	19%	1,2

CHAPITRE 5 DISCUSSION GÉNÉRALE

Ce chapitre présente une discussion générale sur l'implémentation d'ASIP pour algorithmes de reproduction de tons. Comme expliqué précédemment, nous avons réussi à accélérer deux algorithmes à l'aide d'instructions spécialisées, tout en gardant un équilibre avec la surface additionnelle requise. Le premier processeur est basé sur le langage LISA avec l'algorithme global de Reinhard comme cible, et le deuxième ASIP étend le processeur Xtensa de base pour l'algorithme local proposé par Vytla, semblable à celui de Fattal. Nous comparerons les implémentations sous différents points de vue. D'un côté, la section 5.1 les compare en fonction de la qualité des images, de la nature des algorithmes ainsi que de la performance et des méthodologies de conception des ASIP. D'un autre côté, nous présentons à la section 5.2 une comparaison avec d'autres travaux en considérant notamment les critères de performance, de surface et de consommation de puissance. Finalement, la section 5.3 regroupe les limitations du présent travail.

5.1 Comparaison des deux implémentations de reproduction de tons

5.1.1 Qualité des images résultantes

Nous avons montré que les deux ASIP proposés produisent des images à haute qualité. Nous avons comparé les images HDR aux images LDR résultantes en utilisant la métrique DRI (voir les sections 4.2 et 4.3). Selon cette métrique, nous pouvons affirmer que l'algorithme global implémenté ici produit normalement des images plus fidèles aux images HDR que l'algorithme local, car il y a moins de distorsions. Les images produites avec l'algorithme local présentent plus de distorsions dues à l'amplification du contraste qui n'existait pas dans l'image HDR. Cependant, cette distorsion n'est pas une caractéristique nécessairement mauvaise. Par exemple, pour des applications visant l'amélioration de la visibilité routière des conducteurs pendant la nuit, il est souhaitable d'augmenter le contraste dans les zones les plus sombres, bien que ces régions présentent un faible contraste dans l'image de référence. Donc, dans certains cas, cette distorsion devient une propriété désirable de la reproduction de tons.

La distorsion sur le contraste invisible est difficile à juger dans les algorithmes de TM. Sans avoir un écran HDR disponible ou la scène réelle, nous n'avons pas la certitude que la métrique révèle

des distorsions correctes seulement en regardant les images résultantes. Pourtant, les deux autres types de distorsion sont plus intuitifs. Si les images des deux algorithmes sont comparées, les pertes de contraste et les fortes distorsions occasionnées par des artéfacts coïncident avec des évaluations subjectives. C'est le cas des vitraux de l'image « Nave », reproduits encore à la Figure 5.1. Les deux régions rouges dans le plan de Reinhard (voir la Figure 4.7) correspondent aux halos autour des vitraux de l'image résultante et probablement à la saturation des valeurs due à une luminosité élevée. Les détails des vitraux sont plus visibles dans le résultat de l'algorithme local, même si un manque de contraste est observé autour d'eux. Ces dernières remarques équivalent aux distorsions en rouge moins prononcées pour la méthode de Vytla et aux zones vertes qui les entourent, respectivement (voir la Figure 4.11).

a) b)

Figure 5.1: Régions de Nave qui présentent de fortes distorsions avec les algorithmes de a) Reinhard, et de b) Vytla

Nous avons aussi mesuré le PSNR pour comparer avec la métrique DRI. Pour les images « Belgium », « Nave » et « Memorial », les valeurs obtenues ont été les mêmes : 103 dB, 3 dB et 40 dB, respectivement. Cela vérifie que le PSNR n'est pas une métrique fiable pour comparer deux images à plages dynamiques différentes.

En termes de la qualité de l'image, nous ne pouvons pas garantir que les meilleurs résultats soient toujours obtenus par l'algorithme global ou par l'algorithme local. En général, si l'intérêt est d'avoir une bonne qualité d'ensemble, un algorithme global est approprié. Si l'intérêt concerne plutôt des zones en particulier et que la plage des luminances est très élevée, un algorithme local peut être plus utile. Au lieu d'en choisir un, il serait plus adéquat de travailler avec les deux. L'algorithme global serait à la base, car des images à haute qualité peuvent être obtenues à une complexité inférieure. Si l'image HDR est disponible, l'algorithme local s'exécuterait au besoin seulement dans les régions de l'image définies par l'utilisateur ou celles d'une perte critique de

contraste, par exemple. Afin d'estimer ces dernières régions sans qu'une inspection visuelle ne soit nécessaire, cette méthode aurait besoin d'une métrique objective de qualité. Le plan généré par la métrique DRI pourrait servir d'inspiration.

Nous avons constaté la difficulté à évaluer de manière objective la qualité des images générées par des algorithmes de TM. Même si la métrique DRI aide à déterminer certaines distorsions, elle ne fournit pas de scores et sa complexité en termes de calculs est élevée. Cela rend compliquée l'automatisation de l'évaluation, surtout si le but est de l'inclure dans un système embarqué avec contraintes de temps. De plus, lorsque l'image HDR n'est pas disponible, il est ardu de créer une image idéale (*ground truth*) qui guide l'évaluation de la qualité.

5.1.2 Performance

Nous avons développé deux processeurs spécialisés pour deux algorithmes de reproduction de tons. Le premier ASIP est basé sur le modèle LISA du processeur LTRISC avec l'algorithme global de Reinhard comme cible, et le deuxième ASIP étend le processeur Xtensa de base pour accélérer l'algorithme local proposé par Vytla, basé sur celui de Fattal. Nous avons montré que le processeur LTRISC de base prend environ 9 Mcycles pour exécuter l'algorithme global de Reinhard (des détails à la section 4.2.2), alors que l'Xtensa sans accélération consomme autour de 1800 Mcycles pour la solution locale de Vytla (voir la section 4.3.2). Autrement dit, l'algorithme local implémenté est 200 fois plus complexe que l'algorithme global. Augmenter la performance de la méthode locale pose donc plus de défis. Il faut noter que le nombre de cycles a été obtenu pour une image de 256×192 pixels.

Comme on l'a expliqué précédemment, l'algorithme de Vytla travaille sur le domaine logarithmique. Nous avons ajouté une instruction spécialisée pour le calcul du logarithme dans l'algorithme de Reinhard. Donc, nous pouvons analyser l'effet d'inclure une telle instruction dans l'algorithme local. Le facteur d'accélération pour le calcul du logarithme seul (sans tenir compte du reste de l'application) est de 5,65. Dans ce cas, le processeur atteindrait une accélération de 1,2 pour l'algorithme local. Si l'accélération des filtres gaussiens y est rajoutée, la performance serait environ de 1,8 fois plus élevée que la configuration de base.

Dans la section 4.2.2, nous avons montré les résultats de performance du processeur LTRISC spécialisé pour l'algorithme global en termes des taux de trame. En revanche, la performance de

l'ASIP pour l'algorithme local a été présentée selon le nombre de cycles (voir la section 4.3.2). En supposant que la fréquence d'horloge de ce deuxième ASIP ne se voit pas modifiée de manière importante par l'extension au jeu d'instructions, nous pouvons estimer le taux de trame atteint. Le processeur LTRISC proposé traite des images de 256×192 pixels à 25 fps, à une fréquence d'horloge de 85 MHz. Au contraire, le processeur Xtensa particularisé avec les instructions du filtre gaussien et du logarithme atteint des taux inférieurs à 1 fps, à une fréquence d'horloge d'environ 320 MHz. C'est-à-dire, même avec une fréquence d'horloge basse (30% de la fréquence de l'Xtensa), l'algorithme global est exécuté au moins 25 fois plus vite sur le processeur LTRISC que l'algorithme local. Par conséquent, implémenter cet algorithme global sur le processeur Xtensa permettrait d'atteindre des taux plus élevés, supérieurs à 97 fps (en incluant toutes les instructions spécialisées proposées). Pour des images de 512×384 pixels, les taux seraient proches de 25 fps.

5.1.3 Conception d'ASIP

Nous avons exploré les deux tendances actuelles de conception automatisée d'ASIP : en utilisant un langage de description architecturale et avec un processeur de base configurable et extensible. Pour ces deux méthodologies de conception utilisées dans ce travail, nous remarquons que la première phase d'analyse des fonctions critiques est à la fois complexe et importante. Au début, le code de base de l'application n'est pas nécessairement disponible. Donc, il est possible de faire des estimés à partir d'une analyse minutieuse de chaque étape de l'algorithme, basée par exemple sur un graphe de flot de calcul. Pour des algorithmes relativement moins complexes, comme l'algorithme global, cette approche s'avère appropriée. Toutefois, il faut prendre en compte que lorsqu'il y a des opérations arithmétiques comme un logarithme ou une fonction exponentielle, ces opérations sont en général à ce moment-là des « boîtes noires ». Les détails d'implémentation de ce type d'opérations peuvent générer une influence remarquable dans le nombre de cycles.

Comme mentionné à la revue de littérature (section 1.4.2), il est possible de générer automatiquement une description RTL synthétisable du processeur à partir du modèle LISA. C'est une fonctionnalité appréciée. Cependant, il y a plusieurs aspects à améliorer. Par exemple, même si les vecteurs de bits sont très utilisés dans une description matérielle, le générateur du processeur ne permet pas d'effectuer des opérations entre eux. Pour sa part, le simulateur de *Synopsys Processor Designer* permet d'utiliser les vecteurs de bits et d'effecteur des simulations

sans problèmes. De plus, une fois le processeur LTRISC généré, les codes RTL contenaient des erreurs qui empêchaient la synthèse sur le FPGA. Donc, même dans la phase d'exploration d'architectures, nous avons été obligés de modifier plusieurs blocs de code VHDL, dont une division, pour réussir à synthétiser le modèle. En somme, la tâche dite « automatique » est loin d'être parfaite. En tout cas, LISA fournit une flexibilité importante pour la conception d'ASIP.

D'un autre côté, il faut tenir compte de l'exécution de l'application en virgule flottante ou en virgule fixe. Pour atteindre des augmentations majeures dans la performance, il est normalement recommandé d'effectuer les opérations en virgule fixe. Cependant, cette étape peut devenir facilement le goulot d'étranglement du flot de conception si l'algorithme est complexe. Dans le cas de l'algorithme global de reproduction de tons, nous avons pris à peu près 2 semaines pour faire la conversion virgule flottante à virgule fixe. Par contre, le même processus pour l'algorithme local a pris environ 2 mois.

5.2 Comparaisons avec d'autres implémentations

Dans cette section, nous comparons les résultats des deux processeurs spécialisés proposés avec d'autres types d'implémentations trouvées dans la littérature. Cette comparaison est effectuée pour les deux algorithmes de TM cibles, principalement selon deux critères : la performance et la surface requise. Étant donné que la performance de l'algorithme global de Reinhard suit un comportement linéaire en fonction de la taille de l'image, nous pouvons effectuer les comparaisons plus facilement en utilisant le temps normalisé. Ainsi, le temps pour traiter une image avec l'algorithme global sur le processeur LTRISC particularisé est de 0,8 μs/pixel, alors qu'avec le processeur G3 de l'iBook Apple (voir la section 1.1.3) est de 1,9 μs/pixel. Ceci représente une accélération de 2,4× face au processeur G3. Ce résultat est remarquable si on tient compte que la fréquence d'horloge de notre processeur est seulement de 85 MHz, alors que celle du processeur G3 est 800 MHz. Une fréquence d'horloge élevée provoque normalement une plus grande consommation de puissance et génération de chaleur.

D'un autre côté, nous avons pris le code C++ de l'algorithme de Reinhard de la bibliothèque de programmes PFStmo [11]. Pour une implémentation purement logicielle en virgule flottante, le temps normalisé obtenu sur un processeur Intel Core 2 Duo à 1,5 GHz avec 1 GB de RAM est de 0,6 μs/pixel. Cette valeur représente une accélération de 1,3 par rapport au processeur LTRISC.

La valeur de performance obtenue avec le processeur LTRISC est proche, même si la fréquence d'horloge équivaut à 6% de celle du processeur Intel. En conséquence, l'ASIP proposé présente un meilleur compromis entre la performance et la consommation d'énergie qu'un processeur à usage général.

Comme on l'a dit à la section 1.1.3, Goodnight et al. [33] ont implémenté l'algorithme de Reinhard et al. sur un GPU. Ils n'ont pas spécifié les taux de trame atteints pour l'algorithme global. Ils n'ont indiqué que les valeurs étaient très élevées. On peut supposer qu'ils aient atteint au moins 30 fps pour des images de 256×256 pixels, puisqu'ils ont défini le temps réel comme des taux supérieurs à 30 Hz et qu'ils ont obtenu 20 fps pour l'algorithme local de Reinhard (plus complexe que la version globale). Pour cette résolution d'image, le taux de trame résultant avec le processeur LTRISC proposé est environ 19 fps. Cela veut dire que l'application de Goodnight et al. s'exécuterait au moins 1,6 fois plus vite que la nôtre. Cependant, pour avoir utilisé une unité très performante conçue pour des applications graphiques, avec 8 fois plus la puissance du calcul que les processeurs Pentium 4 les plus rapides [33], nous espérions des taux de trame beaucoup plus élevés et aptes au temps réel avec des images à haute résolution (au moins de 1024×768 pixels). En résumé, cette plateforme est gourmande en consommation énergétique et ne satisfait pas les contraintes liées à la consommation de puissance imposées par les systèmes embarqués portables à ressources limitées.

En ce qui concerne l'algorithme local implémenté sur le processeur Xtensa, les seules comparaisons possibles concernent visent le travail original des auteurs. Vytla et al. [25] ont utilisé un FPGA comme plateforme matérielle. Nous nous attendions à ce que leur performance soit nettement plus élevée que la nôtre. Comme mentionné à la revue de littérature, ils traitent une image de 1024×768 pixels à 100 fps. Cependant, ils prennent comme entrée les résultats de la pyramide gaussienne du travail de Hassan [31]. De plus, ils supposent avoir une caméra HDR qui donne le logarithme de la luminance, raison pour laquelle ils ne calculent pas le logarithme. Pour s'ajuster à leurs conditions, nous avons enlevé le logarithme et la pyramide gaussienne de nos calculs du nombre de cycles. Ayant une fréquence d'horloge de 320 MHz, le processeur Xtensa de base effectue la reproduction de tons d'une image de 256×192 pixels à moins de 1 fps. Les différences en performance sont donc de trois ordres de grandeur. Il faut noter que nous ne pouvons que comparer avec le processeur de base sans accélération, puisque les instructions conçues affectent seulement les deux fonctions enlevées.

Tel que décrit à la section 1.1.3, Chiu et al. [5] ont intégré deux algorithmes basés sur la méthode global de Reinhard et l'approche de Fattal. Ils atteignent aussi des taux de trames élevés : 60 fps avec des images de 1024×768 pixels. Ce résultat est possible principalement grâce à un ASIC chargé des parties critiques des deux algorithmes.

Les résultats précédents, notamment pour l'algorithme local, nous amèneraient à penser qu'en principe un ASIP n'est peut-être pas la plateforme idéale pour atteindre des performances élevées pour ce type d'algorithmes. Pour traiter des images de 256×192 pixels à 30 fps avec le processeur Xtensa, il faudrait un facteur d'accélération de 171. Toutefois, des solutions envisageables incluent l'ajout d'un coprocesseur, l'utilisation de multiples processeurs qui permettent un traitement multifilière et l'optimisation du code de base au niveau algorithmique. Dans les deux premiers cas, l'augmentation de la performance peut être atteinte au détriment de la surface supplémentaire requise. D'ailleurs, la flexibilité au niveau de la programmation qu'offrent les ASIP donne de forts avantages quant au temps de mise sur le marché, face aux implémentations dédiées sur un FPGA ou un ASIC.

En termes de l'utilisation de la surface, il n'est pas facile de donner une conclusion définitive. Nos résultats sont en termes de portes logiques pour l'ASIP de l'algorithme local, alors que Vytla et al. rapportent le nombre d'éléments logiques (LE, *Logic Element*) occupés d'un FPGA. Vytla et al. ont signalé que leur design occupe 9019 éléments logiques dans un Stratix II, sans tenir compte de la pyramide gaussienne [25]. Hassan a utilisé 16929 LE, d'un dispositif Stratix bien que pas exactement le même, pour une pyramide gaussienne à 4 niveaux [31]. Comme les LE des deux FPGA Stratix sont équivalents, l'implémentation de la méthode locale utiliserait environ 26000 éléments logiques. Par ailleurs, un LE d'un dispositif APEX est grossièrement comparable à un LE d'un Stratix. Selon Altera, un APEX 20K de 24320 éléments logiques équivaut à 600000 portes logiques [57]. Donc, pour avoir une idée, le système matériel complet de Vytla utiliserait plus de 600000 portes logiques. Ceci impliquerait plus de 500000 portes logiques que le processeur Xtensa particularisé.

Le processeur proposé par Chiu et al. nécessite environ 769000 portes logiques. Dans ce cas, les auteurs ont implémenté l'algorithme global de Reinhard et une version modifiée de l'algorithme de Fattal. À notre connaissance, Xilinx ne fournit pas d'équivalences entre les LUT des Virtex-5 et le nombre de portes logiques. Cependant, Xilinx spécifie une méthodologie pour estimer les

portes logiques équivalentes des FPGA XC5000 [58]. Pour un XC5000, le nombre typique de portes logiques par LUT est environ 12. Chacun de ces LUT contient 4 entrées, alors que les Virtex-5 possèdent des LUT à 6 entrées. En supposant qu'un LUT des Virtex-5 équivaut à 1,3 LUT des XC5000, notre ASIP pour l'algorithme global aurait environ 6600 LUT du type XC5000. Ceci impliquerait près de 80000 portes. Par conséquent, notre processeur occuperait à peu près 180000 portes logiques en incluant les deux algorithmes, une consommation de surface 77% inférieure à celle de Chiu et al.

Une dernière comparaison que nous désirons aborder concerne les descriptions TIE générées automatiquement par les outils de Tensilica. Le Tableau 5.1 regroupe les résultats de performance, de surface additionnelle et une estimation de l'efficacité énergétique de six configurations suggérées par le compilateur XPRES pour l'algorithme de Vytla. Nous remarquons que la première configuration atteint une accélération supérieure à la nôtre, mais à un coût très élevé en surface. Dans ce cas, le nombre de portes logiques supplémentaires est proche de la surface du processeur de base. Au contraire, la cinquième configuration augmente la performance de 20% avec une surface additionnelle négligeable, de même que nos résultats si seulement l'instruction spécialisée du logarithme est ajoutée. De plus, l'accélération atteinte avec les instructions du filtre gaussien est plus grande que celles des TIE automatiques 3 à 6. Cependant, il est intéressant de remarquer que XPRES donne aussi comme possibilité d'améliorer la performance en diminuant un peu la surface du processeur (TIE xpress 6). Quant à la deuxième TIE automatique, celle-ci présente un bon compromis entre performance et surface additionnelle requise. Ses résultats sont semblables aux nôtres, une fois que les instructions du filtre gaussien et du logarithme sont inclues (voir la section 5.1.2).

Tableau 5.1: Résultats des TIE générées par le compilateur XPRES

Configuration du processeur	**Nombre de cycles**	**Facteur d'accélération**	**Surface additionnelle (# portes logiques)**	**Augmentation de la surface**	**Efficacité énergétique**
TIE xpres 1	654.983.793	2,8×	70.593	89%	1,47
TIE xpres 2	1.030.441.979	1,8×	17.818	23%	1,44
TIE xpres 3	1.426.416.946	1,3×	2.715	3%	1,24
TIE xpres 4	1.426.469.212	1,3×	2.302	3%	1,24
TIE xpres 5	1.467.149.304	1,2×	827	1%	1,23
TIE xpres 6	1.489.223.224	1,2×	-346	0%	1,23

Pour chacune des configurations générées par le compilateur XPRES, nous avons aussi calculé le produit AT en divisant le facteur d'accélération par le facteur d'augmentation de surface. Cette métrique fournit une estimation du facteur d'amélioration de la consommation d'énergie. En regardant les valeurs de la dernière colonne du Tableau 5.1, nous remarquons que les valeurs de l'efficacité énergétique des deux premières configurations automatiques se ressemblent. Néanmoins, étant donné le coût matériel de la première configuration, la deuxième semble avoir un meilleur compromis. Par rapport à notre proposition, si les instructions du filtre gaussien et du logarithme sont rajoutées, le résultat d'efficacité énergétique serait au même niveau que la « TIE xpres 2 ». Si l'instruction du logarithme n'est pas considérée, l'amélioration de la consommation d'énergie se rapprocherait des niveaux des configurations 3 à 6.

5.3 Limitations

Certaines limitations ont été observées dans le déroulement de ce projet. Une limitation de l'utilisation des outils de Tensilica est que la fréquence d'horloge du processeur ne considère pas l'extension au jeu d'instructions. C'est pourquoi nous avons basé nos calculs de performance pour l'algorithme local en termes du nombre de cycles. Notre supposition est que les instructions ajoutées n'ont pas modifié de manière importante la fréquence d'horloge, tel qu'avec le processeur LTRISC.

Les résultats du processeur spécialisé conçu avec LISA comportent aussi une limitation. Les nombres de cycles générés par la simulation n'ont pas tenu compte de la hiérarchie de mémoire. Donc, il n'y a pas eu des cycles associés aux échecs causés par des données manquantes dans la mémoire cache. Si une mémoire cache est ajoutée, il faut trouver une configuration de façon à minimiser les échecs. Obtenir cette configuration implique définir la taille de la mémoire, le nombre de lignes et le niveau d'associativité pour que le pourcentage du nombre de cycles associés aux échecs se rapproche de 0%. De cette manière, la différence entre les deux simulations (sans et avec cache), en termes du nombre de cycles, sera négligeable.

Une autre limitation réside dans l'implantation de l'algorithme local avec le langage C. Les codes exécutés sur le processeur Xtensa n'ont pas été optimisés. Par exemple, les opérations qui calculent le filtre gaussien sont toujours effectuées, même entre valeurs nulles. Le cas le plus remarquable se trouve aux limites de l'image.

CONCLUSION

Dans ce travail, nous avons conçu des ASIP pour accélérer deux algorithmes de reproduction de tons (TM), tout en gardant un compromis avec le coût matériel additionnel requis. Concernant le premier objectif, nous avons analysé les deux méthodes choisies en termes de la qualité des images résultantes et des besoins en calculs et mémoire. La première méthode a été la version globale de l'algorithme de Reinhard et al. La deuxième méthode a été l'algorithme local de Vytla et al., une version modifiée de l'algorithme proposé originalement par Fattal et al.

Tout d'abord, nous avons effectué une comparaison de quatre méthodes d'amélioration du contraste d'images LDR, y compris les algorithmes de TM de Reinhard et de Fattal. Selon les résultats expérimentaux, l'algorithme d'Arici obtient le meilleur compromis entre la qualité de l'image et le temps d'exécution pour la surveillance d'animaux de laboratoire. Bien que l'algorithme de Fattal améliore le contraste tout en préservant les détails dans les régions ombragées, ses besoins en calculs ne permettent pas d'atteindre le temps réel avec une implémentation purement logicielle sur le processeur utilisé. Une autre observation au sujet des techniques d'amélioration d'images concerne la robustesse des valeurs des paramètres face aux différentes conditions d'éclairage. Il est connu que ce type de techniques est axé sur les problématiques à résoudre. Cependant, lors des changements des conditions d'éclairage, quelques algorithmes peuvent être plus sensibles que d'autres en termes de la qualité des images résultantes. Le choix des valeurs des paramètres devient donc une étape cruciale et doit être effectué soigneusement.

Des comparaisons en termes de la qualité des images résultantes ont aussi été effectuées dans le contexte de l'imagerie HDR. De manière subjective, l'algorithme global et l'algorithme local implémentés produisent des images à haute qualité. De manière objective, nous avons comparé les images HDR aux images LDR résultantes en utilisant le plan de distorsions généré par la métrique DRI. Selon cette métrique, l'algorithme global produit normalement des images plus fidèles aux images HDR que l'algorithme local, car les distorsions sont moins présentes. Cependant, il est difficile à garantir que les meilleurs résultats soient toujours obtenus avec un algorithme unique.

Le premier objectif de ce travail concernait aussi l'analyse des besoins en calculs d'algorithmes de TM. Pour les deux méthodes choisies, nous avons estimé le nombre d'exécutions de chaque

opération à l'aide de graphes de flot de calcul. Nous avons remarqué les différences en calculs requis entre l'algorithme global et l'algorithme local. Dans ce cas, l'algorithme local de Vytla implémenté est environ 200 fois plus complexe que l'algorithme global de Reinhard. Dans ces deux méthodes, les opérations qui posent plus de défis au moment de l'implémentation en virgule fixe sont le logarithme et sa fonction inverse.

Le deuxième objectif portait sur l'implémentation des processeurs spécialisés afin d'augmenter la performance d'algorithmes de reproduction de tons. Une implémentation de l'algorithme global de Reinhard et al. a été développée sur un processeur LTRISC spécialisé. Nous avons ajouté trois instructions à l'aide du langage de description architecturale LISA afin d'accélérer cette application, tout en offrant une flexibilité proche des processeurs à usage général. Le processeur proposé atteint une accélération de 2,7 avec un coût matériel supplémentaire de 22%. L'instruction spécialisée utilisée pour le calcul du logarithme offre une amélioration de la performance de 2,4 avec seulement 4% en augmentation de la surface, par rapport au processeur de base. Le produit AT obtenu indique que le facteur d'amélioration en consommation énergétique est d'environ 2,3. En comparant avec un travail antérieur, nos résultats démontrent que l'ASIP proposé obtient des performances plus élevées qu'avec un processeur à usage général et à plus faible consommation de puissance.

Un autre processeur spécialisé a été développé en étendant le jeu d'instructions d'un processeur Xtensa. Cet ASIP avait pour but l'accélération de l'algorithme local de Vytla et al. Grâce aux instructions spécialisées pour calculer les filtres gaussiens, nous avons obtenu un facteur d'accélération de 1,5 avec une surface additionnelle de 19%. Si des instructions pour le calcul du logarithme y sont aussi ajoutées, le processeur atteint une augmentation de la performance de 1,8. De plus, l'amélioration estimée de la consommation d'énergie est de 1,2. Par ailleurs, nous avons obtenu une accélération de 39 en comparant l'implémentation en virgule fixe et les instructions spécialisées face à l'implémentation en virgule flottante sur le processeur de base.

Quant au troisième objectif, nous avons accéléré le calcul de la pyramide gaussienne modifiée. Pour l'accélérer, nous avons proposé d'utiliser principalement une fusion d'opérateurs, des instructions SIMD et une réutilisation de données. Ces techniques ont permis d'atteindre une accélération de 12,3 en termes du nombre de cycles, par rapport au processeur de base.

D’autres études peuvent être envisagées dans ce domaine, comme celles-ci :

- Un premier aspect porte sur le choix des instructions spécialisées à ajouter au processeur de base. Nous avions plusieurs idées pour accélérer le filtre gaussien, mais nous avons seulement implémenté une à cause des échéanciers. La situation idéale serait d’avoir une métrique d’accélération permettant d’estimer les gains en performance sans avoir besoin d’implémenter les instructions. Ce potentiel d’accélération faciliterait l’exploration théorique de diverses architectures afin d’en choisir la plus appropriée à implémenter.

- Un autre point important consiste à déterminer le type d’architecture de la pyramide gaussienne. Dans ce travail, le calcul de base a été effectué avec un seul noyau et plusieurs itérations. Nous aurions pu aussi utiliser différents noyaux, comparer les complexités, les besoins en mémoire et choisir l’architecture la plus adéquate dans le contexte des systèmes embarqués. Ceci pourrait mener à la proposition d’une méthodologie de sélection d’architectures pour le filtre gaussien, selon les contraintes visées en performance et en mémoire.

- Une dernière remarque concerne l’évaluation objective des algorithmes de TM en termes de la qualité des images. Nous avons constaté la difficulté à juger la qualité des images, notamment de manière objective. Ce sujet pourrait motiver de futurs projets dans le cadre d’une recherche multidisciplinaire, étant donné les différents concepts du système visuel humain à considérer. L’importance d’une métrique de qualité objective réside dans la possibilité d’automatiser l’évaluation en choisissant le meilleur algorithme pour une condition d’éclairage ou un type de scène donné. Elle serait aussi utile dans une approche de TM hybride, capable d’exécuter un algorithme local dans les régions où l’algorithme global ne préserve pas le contraste désiré.

RÉFÉRENCES

[1] E. Reinhard, W. Heidrich, P. Debevec, S. Pattanaik, G. Ward, et K. Myszkowski, *High Dynamic Range Imaging: Acquisition, Display, and Image-Based Lighting*, 2e éd., Burlington: Morgan Kaufmann, 2010.

[2] R. Fattal, D. Lischinski, et M. Werman, "Gradient domain high dynamic range compression," in *SIGGRAPH '02: 29th International Conference on Computer Graphics and Interactive Techniques, 21-26 July 2002*, *USA*, vol. 21, 2002, pp. 249-56.

[3] E. Reinhard, M. Stark, P. Shirley, et J. Ferwerda, "Photographic tone reproduction for digital images," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 267-276, 2002.

[4] A. El-Mahdy et H. El-Shishiny, "High-quality HDR rendering technologies for emerging applications," *IBM Journal of Research and Development*, vol. 54, no. 6, pp. 8:1-8:15, 2010.

[5] C. T. Chiu, T. H. Wang, W. M. Ke, C. Y. Chuang, J. S. Huang, W. S. Wong, R. S. Tsay, et C. J. Wu, "Real-Time Tone-Mapping Processor with Integrated Photographic and Gradient Compression using 0.13 mu m Technology on an Arm Soc Platform," *Journal of Signal Processing Systems for Signal Image and Video Technology*, vol. 64, no. 1, pp. 93-107, 2011.

[6] K. Karuri et R. Leupers, *Application Analysis Tools for ASIP Design Application Profiling and Instruction-set Customization*, New York: Springer, 2011.

[7] S. Saponara, M. Casula, et L. Fanucci, "ASIP-based reconfigurable architectures for power-efficient and real-time image/video processing," *Journal of Real-Time Image Processing*, vol. 3, no. 3, pp. 201-216, 2008.

[8] W. C. Kao, M. C. Hsu, et Y. Y. Yang, "Local contrast enhancement and adaptive feature extraction for illumination-invariant face recognition," *Pattern Recognition*, vol. 43, no. 5, pp. 1736-1747, 2010.

[9] D. C. Gil, R. Farah, J. M. P. Langlois, G. A. Bilodeau, et Y. Savaria, "Comparative analysis of contrast enhancement algorithms in surveillance imaging," in *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*, 2011, pp. 849-852.

[10] S. Vakili, D. C. Gil, J. M. P. Langlois, Y. Savaria, et G. Bois, "Customized embedded processor design for global photographic tone mapping," in *Electronics, Circuits and Systems (ICECS), 2011 18th IEEE International Conference on*, 2011, pp. 382-385.

[11] G. Krawczyk, "PFStmo:: tone mapping operators." [En ligne]. Disponible: http://www.mpi-inf.mpg.de/resources/tmo/. [Consulté le 10 août 2010].

[12] M. Fairchild, "The HDR Photographic Survey." [En ligne]. Disponible: http://www.cis.rit.edu/fairchild/HDR.html. [Consulté le 27 janvier 2012].

[13] R. Fattal, D. Lischinski, et M. Werman, "Gradient Domain High Dynamic Range Compression." [En ligne]. Disponible: http://www.cs.huji.ac.il/~danix/hdr/. [Consulté le 10 août 2010].

[14] K. Myszkowski, R. Mantiuk, et G. Krawczyk, *High Dynamic Range Video*: Morgan & Claypool Publishers, 2008.

[15] J. Lee, G. Jeon, et J. Jeong, "Piecewise tone reproduction for high dynamic range imaging," *Consumer Electronics, IEEE Transactions on*, vol. 55, no. 2, pp. 911-918, 2009.

[16] A. M. Sa, *High Dynamic Range Image Reconstruction*: Morgan & Claypool Publishers 2008.

[17] G. Yourganov et W. Stuerzlinger, "Tone-Mapping for High Dynamic Range Images," York University, Toronto, Rapport technique, 2001.

[18] G. Ward Larson, H. Rushmeier, et C. Piatko, "A visibility matching tone reproduction operator for high dynamic range scenes," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 3, no. 4, pp. 291-306, 1997.

[19] M. Čadík, M. Wimmer, L. Neumann, et A. Artusi, "Evaluation of HDR tone mapping methods using essential perceptual attributes," *Computers & Graphics*, vol. 32, no. 3, pp. 330-349, 2008.

[20] J. Tumblin et H. Rushmeier, "Tone Reproduction for Realistic Images," *IEEE Computer Graphics and Applications*, vol. 13, no. 6, pp. 42-48, 1993.

[21] K. Matkovic, L. Neumann, et W. Purgathofer, " A Survey of Tone Mapping Techniques," Institute of Computer Graphics and Algorithms, Vienna University of Technology, Rapport technique, 1997.

[22] G. Ward, "A contrast-based scalefactor for luminance display," in *Graphics gems IV*: Academic Press Professional, Inc., 1994, pp. 415-421.

[23] International Commission on Illumination., *An Analytic model for describing the influence of lighting parameters upon visual performance*, vol. 1: Technical Foundations, Paris, France: Bureau central de la CIE, 1981.

[24] B. Horn, "Determining lightness from an image," *Computer Graphics and Image Processing*, vol. 3, no. 4, pp. 277-299, 1974.

[25] L. Vytla, F. Hassan, et J. E. Carletta, "A real-time implementation of gradient domain high dynamic range compression using a local Poisson solver " *Journal of Real-Time Image Processing*, pp. 15, 2011.

[26] F. Durand et J. Dorsey, "Fast bilateral filtering for the display of high-dynamic-range images," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 257-266, 2002.

[27] M. Ashikhmin, "A tone mapping algorithm for high contrast images," in Proceedings of the 13th Eurographics workshop on Rendering, vol., Ed.^Eds., ed. Pisa, Italy: Eurographics Association, 2002, pp. 145-156.

[28] J. Duan, M. Bressan, C. Dance, et G. Qiu, "Tone-mapping high dynamic range images by novel histogram adjustment," *Pattern Recognition*, vol. 43, no. 5, pp. 1847-1862, 2010.

[29] Q. Shan, J. Jiaya, et M. S. Brown, "Globally Optimized Linear Windowed Tone Mapping," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 16, no. 4, pp. 663-675, 2010.

[30] R. C. Gonzalez et R. E. Woods, *Digital Image Processing*, 3e éd., New Jersey: Pearson Prentice Hall, 2008.

[31] F. Hassan, "Real-Time Embedded Algorithms for Local Tone Mapping of High Dynamic Range Images," Ph.D., University of Akron, OH, USA, 2007. [En ligne]. Disponible: http://rave.ohiolink.edu/etdc/view?acc_num=akron1195664951. [Consulté le 27 janvier 2012].

[32] E. Reinhard, G. Ward, S. Pattanaik, et P. Debevec, *High Dynamic Range Imaging: Acquisition, Display, and Image-Based Lighting*, San Francisco: Morgan Kaufmann Publishers, 2006.

[33] N. Goodnight, R. Wang, C. Woolley, et G. Humphreys, "Interactive time-dependent tone mapping using programmable graphics hardware," in Proceedings of the 14th Eurographics workshop on Rendering, Leuven, Belgium: Eurographics Association, 2003, pp. 26-37.

[34] Q. Chen, X. Xu, Q. S. Sun, et D. S. Xia, "A solution to the deficiencies of image enhancement," *Signal Processing*, vol. 90, no. 1, pp. 44-56, 2010.

[35] T. Arici, S. Dikbas, et Y. Altunbasak, "A Histogram Modification Framework and Its Application for Image Contrast Enhancement," *Image Processing, IEEE Transactions on*, vol. 18, no. 9, pp. 1921-1935, 2009.

[36] K. A. Panetta, E. J. Wharton, et S. S. Agaian, "Human Visual System-Based Image Enhancement and Logarithmic Contrast Measure," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 38, no. 1, pp. 174-188, 2008.

[37] S. Lee, "An Efficient Content-Based Image Enhancement in the Compressed Domain Using Retinex Theory," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 17, no. 2, pp. 199-213, 2007.

[38] Z. Wang et A. C. Bovik, *Modern Image Quality Assessment*, USA: Morgan & Claypool Publishers, 2006.

[39] Z. Wang, A. C. Bovik, H. R. Sheikh, et E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *Image Processing, IEEE Transactions on*, vol. 13, no. 4, pp. 600-612, 2004.

[40] S.-D. Chen et A. R. Ramli, "Minimum mean brightness error bi-histogram equalization in contrast enhancement," *Consumer Electronics, IEEE Transactions on*, vol. 49, no. 4, pp. 1310-1319, 2003.

[41] P. Ledda, A. Chalmers, T. Troscianko, et H. Seetzen, "Evaluation of tone mapping operators using a High Dynamic Range display," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 640-648, 2005.

[42] M. Ashikhmin et J. Goyal, "A reality check for tone-mapping operators," *ACM Trans. Appl. Percept.*, vol. 3, no. 4, pp. 399-411, 2006.

[43] T. O. Aydin, R. Mantiuk, K. Myszkowski, et H.-P. Seidel, "Dynamic range independent image quality assessment," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 1-10, 2008.

[44] R. Mantiuk, S. Daly, K. Myszkowski, et H. P. Seidel, "Predicting visible differences in high dynamic range images - Model and its calibration," *Human Vision and Electronic Imaging X*, vol. 5666, pp. 204-214, 2005.

[45] S. Rajotte, D. Carolina Gil, et J. M. P. Langlois, "Combining ISA extensions and subsetting for improved ASIP performance and cost," in *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*, 2011, pp. 653-656.

[46] P. Aubertin, J. M. P. Langlois, et Y. Savaria, "Real-Time Computation of Local Neighborhood Functions in Application-Specific Instruction-Set Processors," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1-13, 2011.

[47] O. Schliebusch, H. Meyr, et R. Leupers, *Optimized ASIP Synthesis from Architecture Description Language Models*, The Netherlands: Springer, 2007.

[48] P. Mishra et N. Dutt, "Architecture Description Languages," in *Customizable Embedded Processors*: Morgan Kaufmann, 2007, pp. 62 - 79.

[49] S. Pees, A. Hoffmann, V. Zivojnovic, et H. Meyr, "LISA - machine description language for cycle-accurate models of programmable DSP architectures," in *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, *New Orleans, USA*, 1999, pp. 933-938.

[50] Tensilica, *Xtensa® LX2 Microprocessor Data Book*, Santa Clara: Tensilica Technical Publications, 2007.

[51] J. N. Mitchell, "Computer Multiplication and Division Using Binary Logarithms," *Electronic Computers, IRE Transactions on*, vol. EC-11, no. 4, pp. 512-517, 1962.

[52] S. Paul, N. Jayakumar, et S. P. Khatri, "A Fast Hardware Approach for Approximate, Efficient Logarithm and Antilogarithm Computations," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 17, no. 2, pp. 269-277, 2009.

[53] B. S. Clark, "Time lapse HDR: time lapse photography with high dynamic range images," M.Sc., Texas A&M University, Texas, USA, 2005. [En ligne]. Disponible: http://hdl.handle.net/1969.1/2408. [Consulté le 2 février 2012].

[54] T. Aydin, "Image Quality Assessment Online: Dynamic Range (In)dependent Metrics Online." [En ligne]. Disponible: http://driiqm.mpi-inf.mpg.de/. [Consulté le 2 février 2012].

[55] P. Debevec, "Recovering High Dynamic Range Radiance Maps from Photographs." [En ligne]. Disponible: http://ict.debevec.org/~debevec/Research/HDR/. [Consulté le 27 janvier 2012].

[56] J. Tumblin et G. Turk, "LCIS: a boundary hierarchy for detail-preserving contrast reduction," in Proceedings of the 26th annual conference on Computer graphics and interactive techniques, ed.: ACM Press/Addison-Wesley Publishing Co., 1999, pp. 83-90.

[57] Altera, "Gate Counting Methodology for APEX 20K Devices," Rapport technique (Application Note) 110, 1999.

[58] Xilinx, "Gate Count Capacity Metrics for FPGAs," Rapport technique (Application Note) XAPP 059, 1997.

[59] H. Keding, "Killing the Pain of Fixed-Point Design." [Webinar de Synopsys]. Disponible: http://www.synopsys.com/Systems/BlockDesign/DigitalSignalProcessing/Pages/Algorithm-eUpdate-Nov2010.aspx. [Consulté le 29 juin 2011].

[60] S. Kim, K.-I. Kum, et W. Sung, "Fixed-point optimization utility for C and C++ based digital signal processing programs," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 45, no. 11, pp. 1455-1464, 1998.

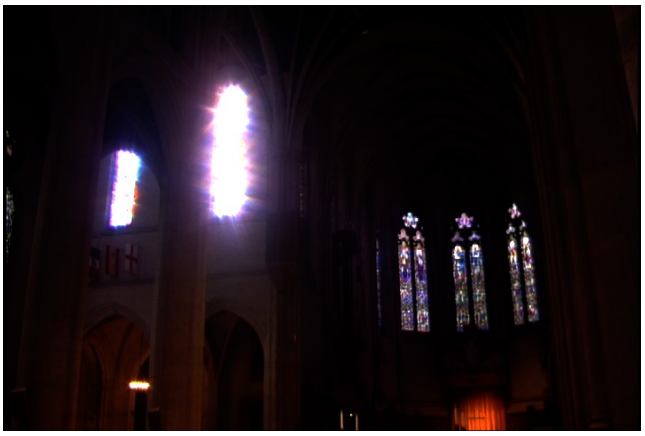
[61] J. Lemieux, "Fixed-point math in C." [En ligne]. Disponible: http://www.eetimes.com/discussion/other/4024639/Fixed-point-math-in-C. [Consulté le 18 juillet 2011].

[62] S. Rein et M. Reisslein, "Low-Memory Wavelet Transforms for Wireless Sensor Networks: A Tutorial," *Communications Surveys & Tutorials, IEEE*, vol. 13, no. 2, pp. 291-307, 2011.

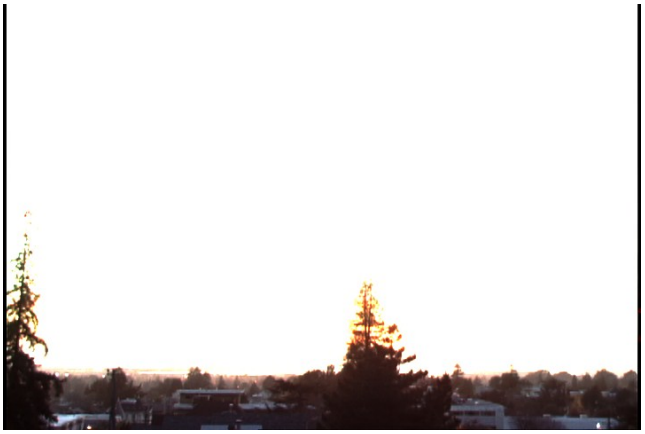
ANNEXE 1 – Images HDR utilisées

Les images montrées ci-dessous correspondent aux images HDR originales sans reproduction de tons, telles qu'affichées sur un écran traditionnel.

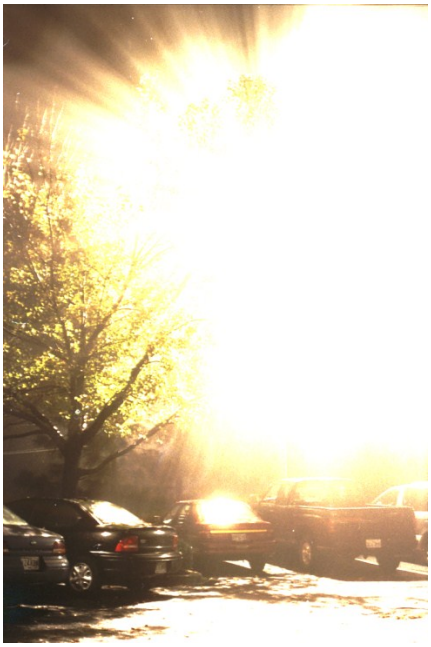
Belgium House



Nave



Vine Sunset



BigFogMap

Memorial

Atrium Night

ANNEXE 2 – Conversion de virgule flottante à virgule fixe

Les premiers codes développés dans ces travaux pour les algorithmes de reproduction de tons de Reinhard et de Vytla/Fattal, étaient en virgule flottante. Afin d'augmenter la performance, nous les avons transformés en virgule fixe à l'aide des simulations sur MATLAB. C'est une étape cruciale, étant donné que la qualité de l'image résultante dépend du nombre de bits choisi. C'est aussi une étape complexe, puisqu'il y a de nombreuses variables qui ont besoin de différentes précisions. De plus, on ne connaît que le nombre total de bits pour l'image originale (32) et pour l'image résultante (8). Donc, comment déterminer le nombre de bits entiers et fractionnaires pour chaque variable ? Nous avons suivi une méthodologie basée sur les étapes présentées ci-dessous [59]:

1. **Spécialisation de l'algorithme**

Cette étape consiste à remplacer les parties idéales de l'algorithme par les approximations respectives. Par exemple, le filtre gaussien utilisé dans l'algorithme de Vytla et Fattal a été calculé comme une convolution avec un noyau 3×3. Nous n'avons pas utilisé la fonction gaussienne originale ni de filtrage dans le domaine des fréquences pour l'application finale. Autrement dit, il faut avoir décidé les détails d'implémentation de l'application finale. De cette façon, il ne sera pas nécessaire de réaliser deux fois la conversion en virgule fixe, et les approximations seront normalement plus faciles à transformer. Dans cet ordre d'idées, les parties génériques du code qui ne sont pas nécessaires devraient être enlevées, par exemple, si des constantes sont déjà connues.

2. **Identification des objets clefs**

La deuxième étape vise à déterminer quelles sont les variables les plus importantes à transformer. Dans le cas des algorithmes de reproduction de tons, il est important de savoir le nombre de bits entiers et fractionnaires de l'image HDR d'entrée. D'autres objets d'intérêt typiques sont les accumulateurs, les variables d'entrée et de sortie de chaque bloc, etc. Les objets non importants sont ceux issus des objets clefs, comme les variables temporelles.

3. **Analyse de données des objets clefs**

Cette étape a pour but de connaître le nombre de bits de la partie entière (IWL) et le signe des objets clefs. Pour ce faire, il faut collecter des données en simulant l'application originale en

virgule flottante. Les simulations doivent être exécutées avec des stimuli suffisants et pertinents. Une estimation du nombre de bits entiers peut être obtenue à partir de la plage de valeurs de la variable d'intérêt. Nous avons appliqué l'équation suivante :

$$IWL = \left\lceil \log_2\left(max(|\mu| + \mathrm{k} \times \sigma, O^{max})\right)\right\rceil,$$

où $\lceil \cdot \rceil$ est la fonction plafond, μ est la moyenne de l'opérande *O,* σ est son écart-type, O^{max} est la valeur maximale de l'opérande et k est une constante égale à 4 [60].

4. Exploration des besoins en précision des objets clefs

L'objectif de la quatrième étape est de déterminer la précision suffisante pour chaque variable d'intérêt, c'est-à-dire, la longueur de mot (WL). L'idée est d'effectuer des simulations avec différents WL et d'explorer l'effet sur la qualité de l'algorithme. Nous avons utilisé des métriques objectives, le MSSIM et le PSNR, pour avoir un indice de la qualité de l'image LDR résultante. Dans ce cas, l'image de référence était celle produite par la simulation en virgule flottante.

Afin de diminuer l'espace de solutions possibles pour WL de chaque variable, nous avons ciblé une résolution ou une granularité en observant les valeurs maximales et les valeurs minimales collectées à l'étape 3. Par exemple, si la plage d'une variable est entre 0 et 1 et nous définissons une résolution *R* de 0,01, il est possible d'avoir une idée du nombre de bits fractionnaires (FWL) de façon à satisfaire l'équation suivante [61] :

$$\frac{1}{2^{FWL}} \geq R \geq \frac{1}{2^{FWL-1}}.$$

Pour cet exemple, FWL est 7, puisque $1/2^7 = 0{,}0078125$.

5. Calcul du format en virgule fixe pour le reste d'objets

Finalement, pour le reste d'objets, il est possible de dériver le format en virgule fixe à partir des objets déjà transformés en suivant quelques règles, comme celles décrites dans la référence [62]. Par exemple, le nombre de bits entiers (IWL) et le nombre de bits fractionnaires (FWL) du résultat d'une somme peuvent être estimés comme suit :

$$a \pm b = c \rightarrow \begin{matrix} IWL(c) = max\left(IWL(a), IWL(b)\right) + 1 \\ FWL(c) = max\left(FWL(a), FWL(b)\right) \end{matrix}.$$

Une autre possibilité est de réaliser la même procédure utilisée pour les objets clefs.